THE BIOROBOTICS INSTITUTE

Scuola Superiore Sant'Anna

Neuromorphic computing

Robotics
M.Sc. programme in Computer Science

Lorenzo Vannucci
l.vannucci@sssup.it

April 21st, 2016

Outline

1. Introduction
2. Fundamentals of neuroscience
3. Simulating the brain
4. Software and hardware simulations
5. Robotic applications

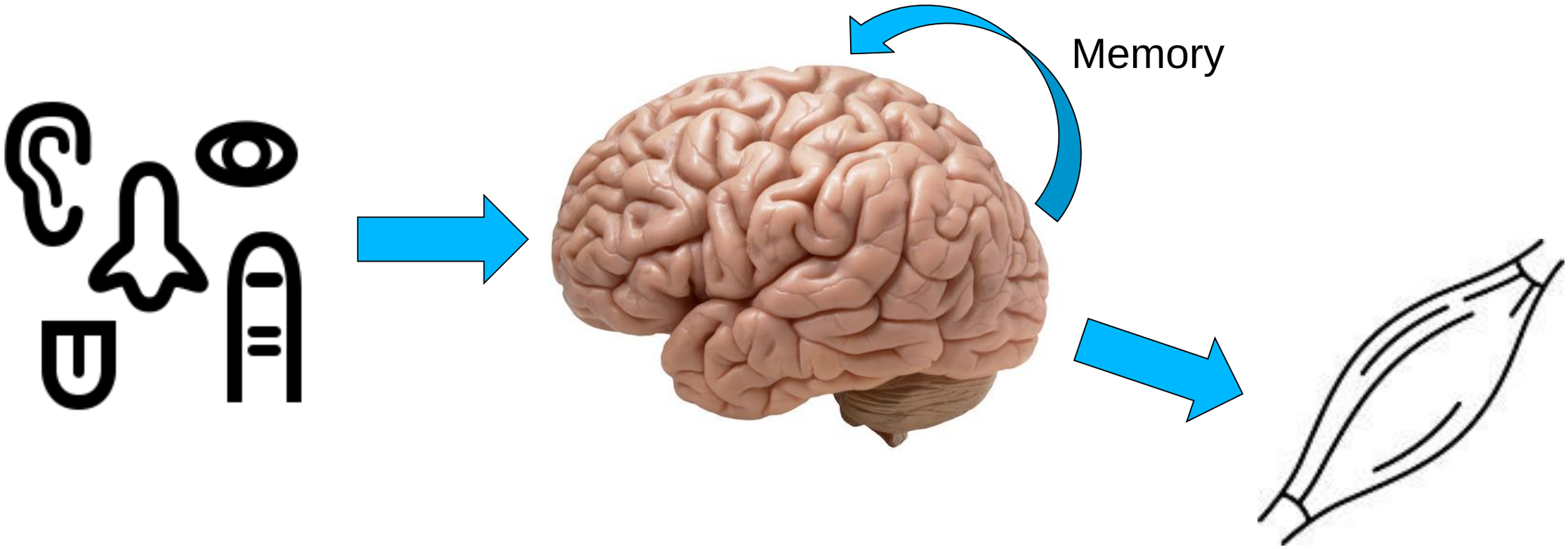
Outline

1. **Introduction**
2. Fundamentals of neuroscience
3. Simulating the brain
4. Software and hardware simulations
5. Robotic applications

What is neuromorphic computing?

We can define **neuromorphic computing** as the act of performing a computation in a manner similar to the brain.

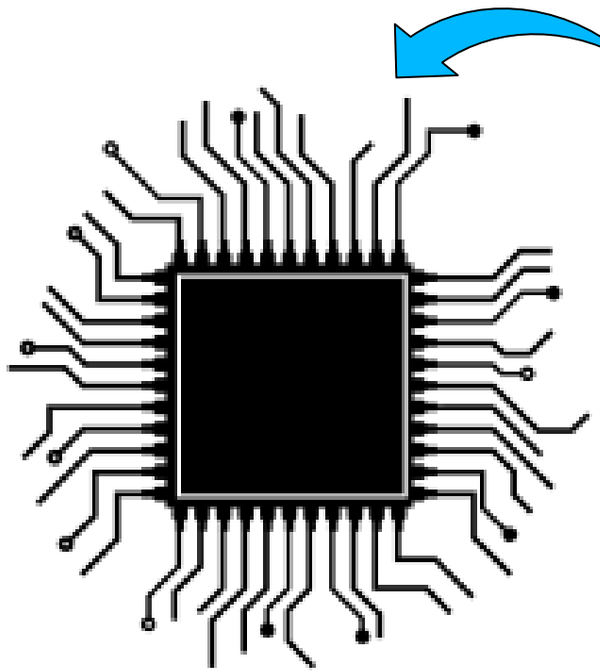
Our brain elaborates inputs coming from our sensors and produces outputs in term of generated motions and stored information.



Why neuromorphic computing (in robotics)?

This kind of computing is very similar to what can be found in a robotic controller.

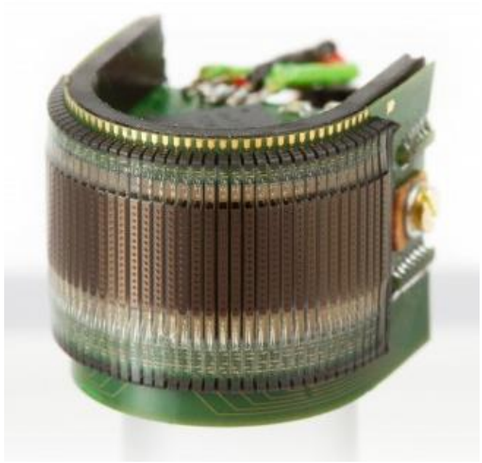
But the sensors and actuators are completely different, compared to humans and animals, thus the brain is substituted by a computer.

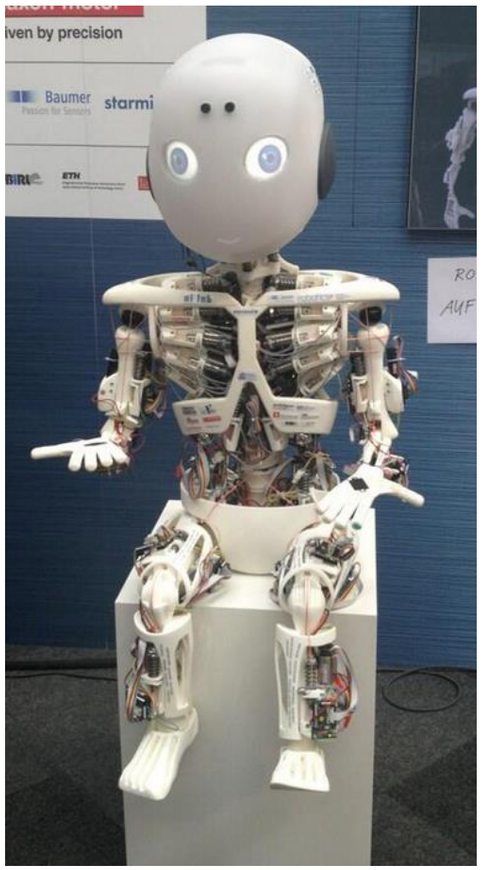


Memory

Why neuromorphic computing (in robotics)?

Today, bio-inspired sensing and actuation technologies are starting to emerge.





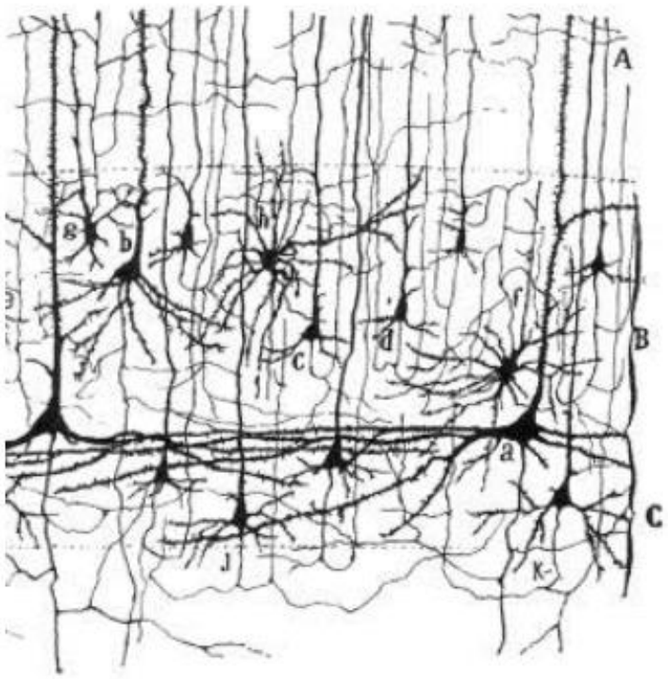
As such, neuromorphic computing can be used to control this new kind of robot.

Outline

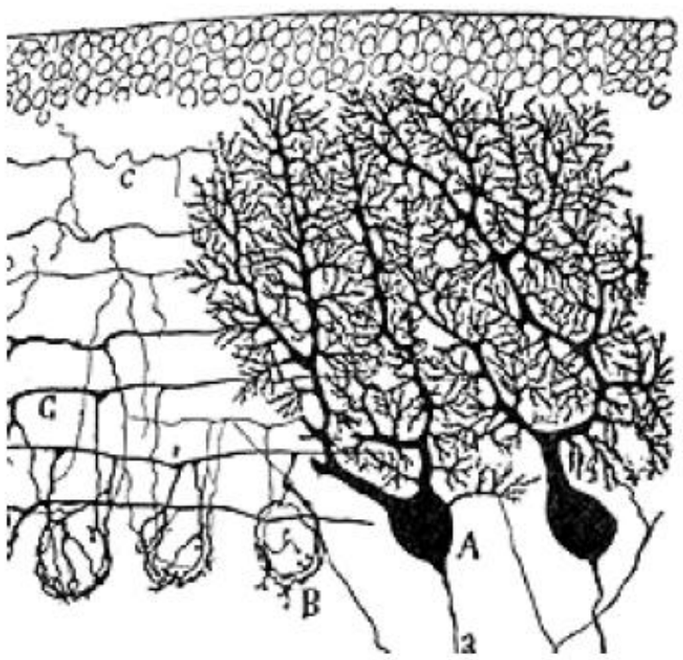
1. Introduction
2. **Fundamentals of neuroscience**
3. Simulating the brain
4. Software and hardware simulations
5. Robotic applications

Neuronal physiology

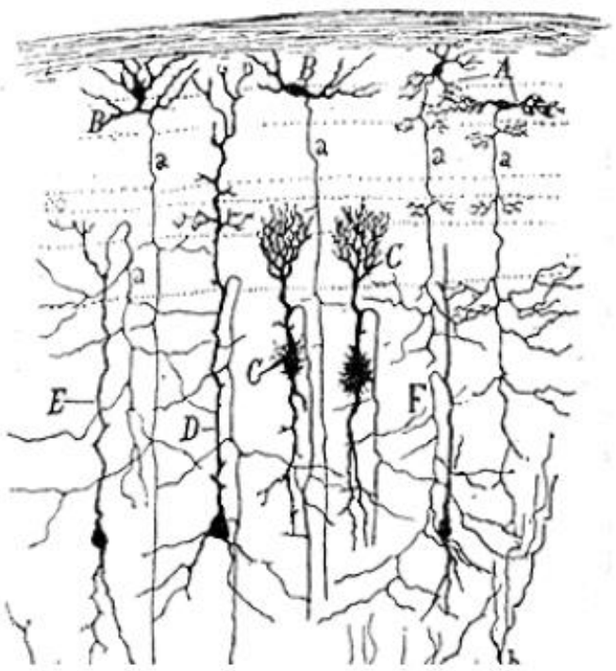
The neuron is the fundamental structural and functional unit of the brain.



Visual Cortex



Cerebellum

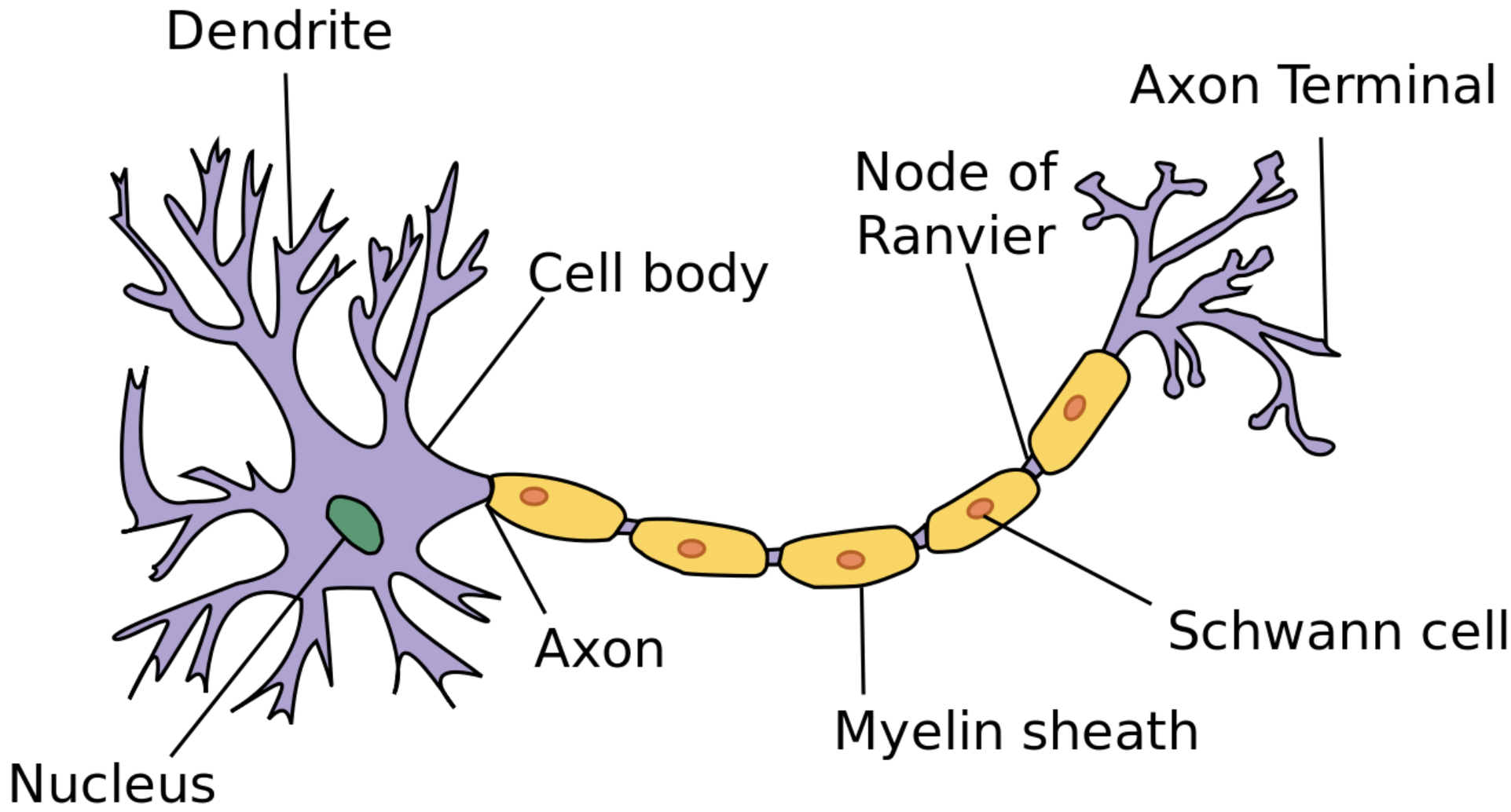


Optic Tectum

(Drawings by Ramón y Cajal, c. 1900)

Neuronal physiology

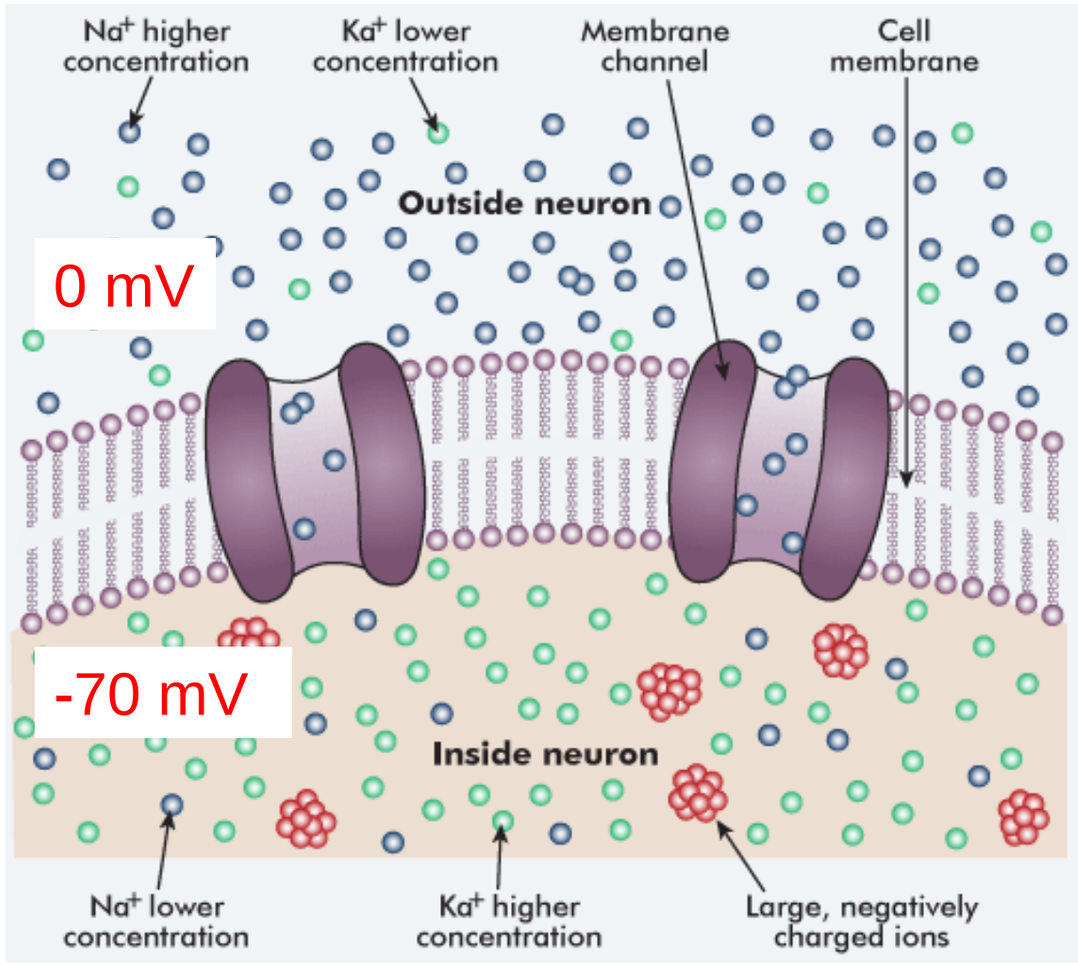
Many kind of neurons share the same cellular physiology.



Neuronal physiology

Neuronal electrophysiological activity lies on the cell membrane.

- Lipid bilayer, impermeable to charged ions.
- Ionic channels allow ions to flow in or out, selectively.
- The neuron maintains a **potential difference** across it membrane via the ionic pumps (expelling Na^+ and allowing K^+ in).
- When no external stimulus is present, we can refer to it as **resting potential**.



Action potentials

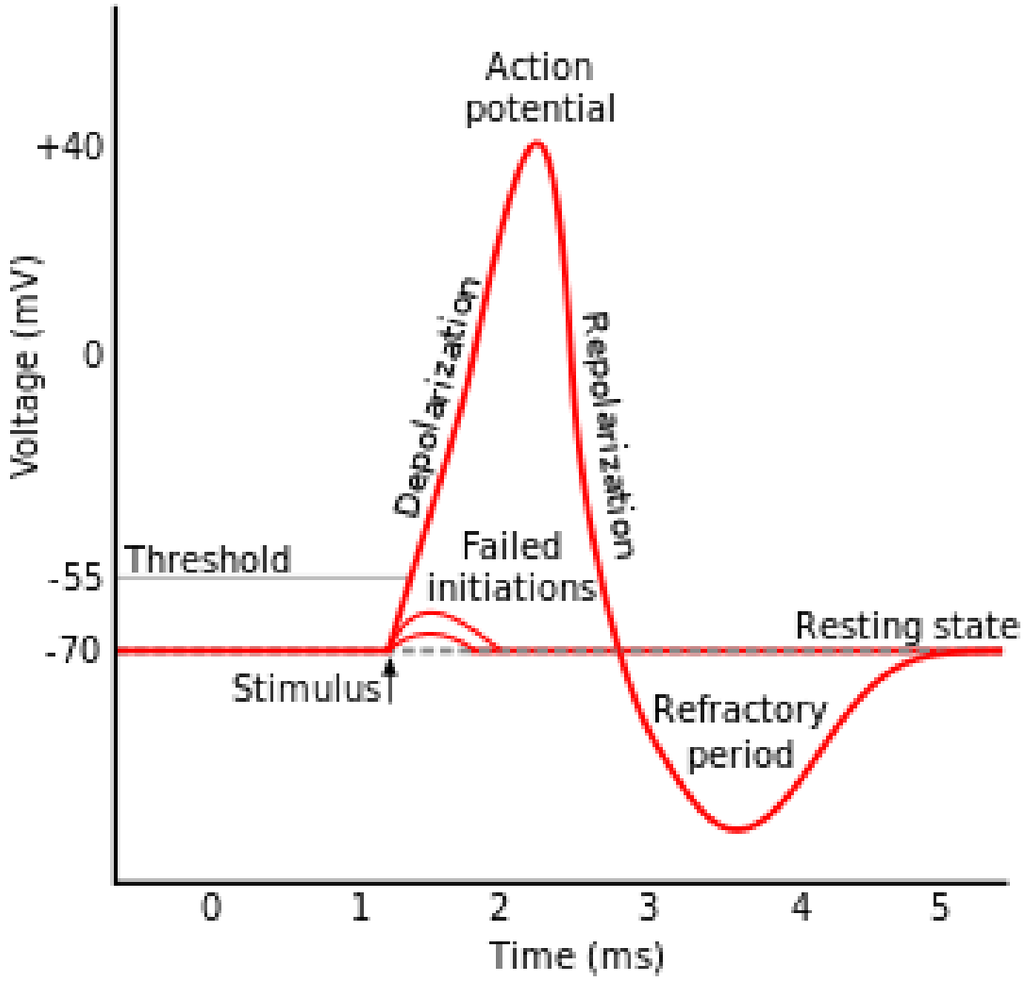
The activity of a neuron (its “output”) is the **action potential** (or spike), generated by voltage-gated ionic channels.

1. An external electric stimulus reach the membrane, depolarizing it.

2. Depolarization of the membrane opens Na^+ channels (→ even more depolarization).

3. If membrane potential exceeds the **threshold potential**, an action potential occurs.

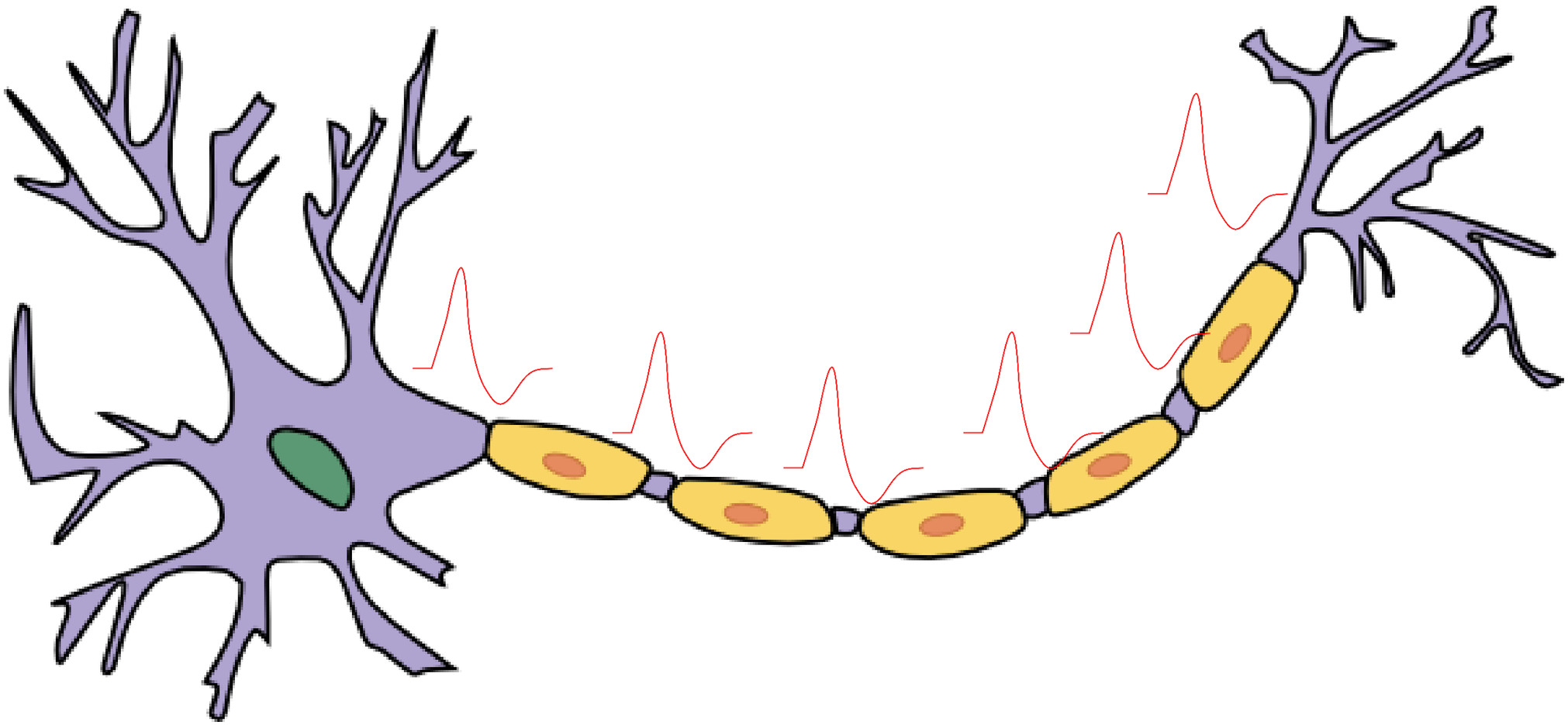
4. Afterwards, the membrane repolarize by expelling K^+ ions and the neuron enters the refractory period.



Action potentials

The action potential is transmitted through the axon towards other neurons.

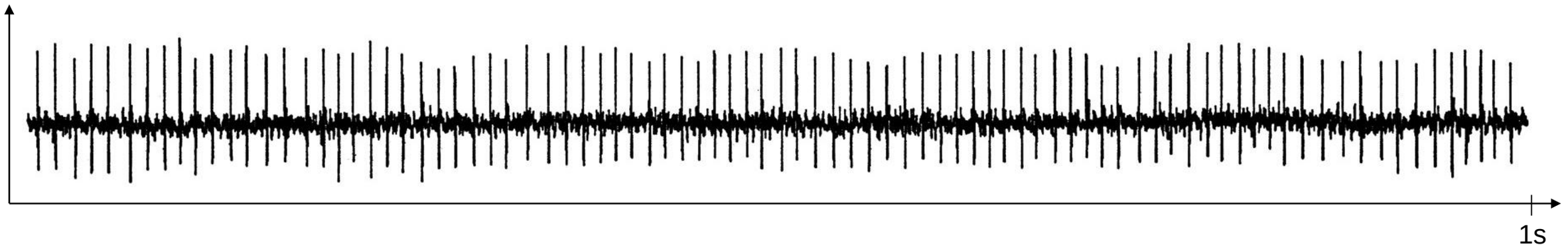
Each non-myelinated section (node of Ranvier) replicates the spike.



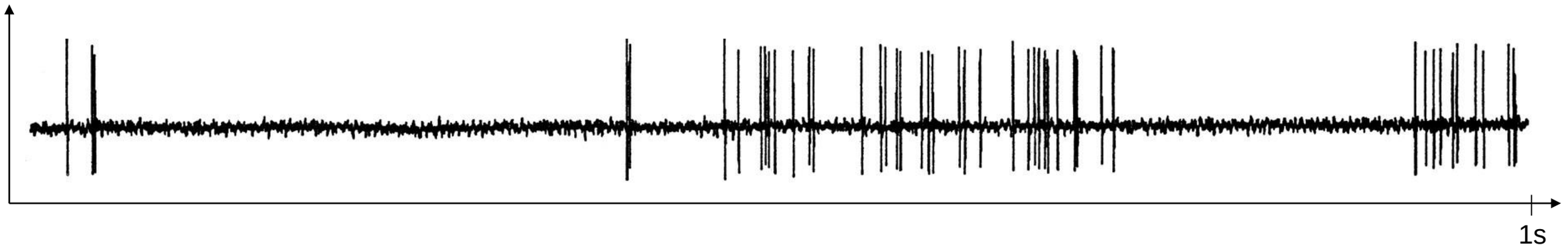
Propagation speed ranges from 1 to 100 m/s.

Action potentials

The activity of a neuron is measured by computing its **firing rate**, expressed as the mean number of spikes per second.



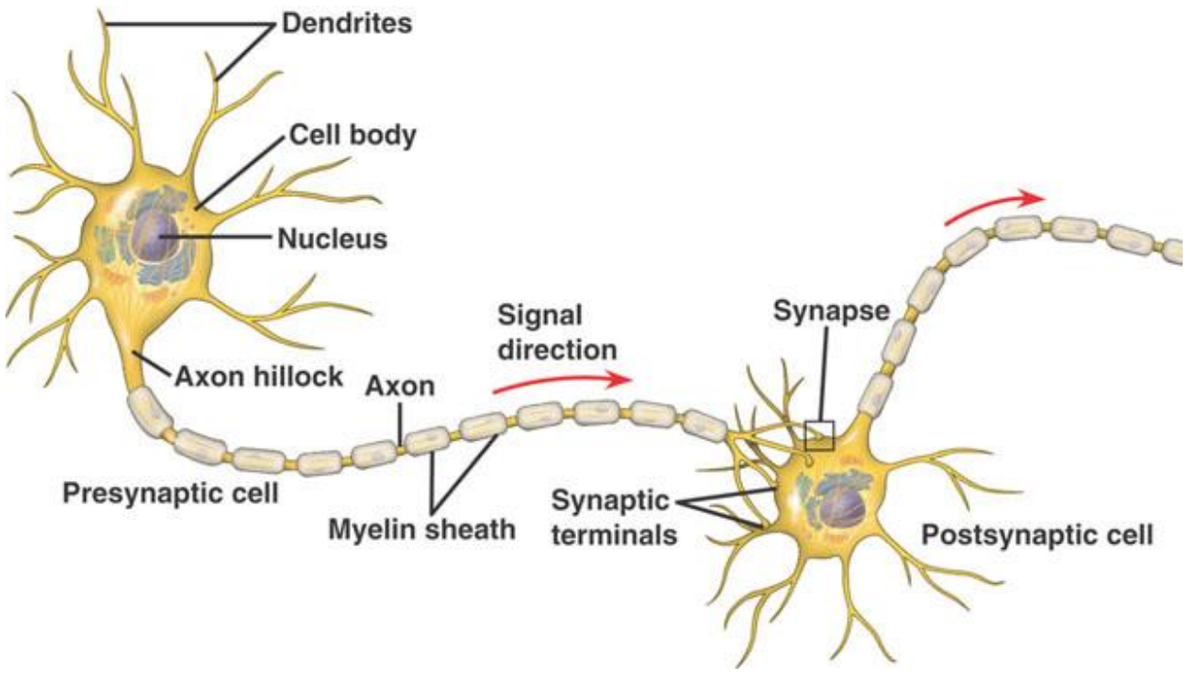
It is not always an easy task!

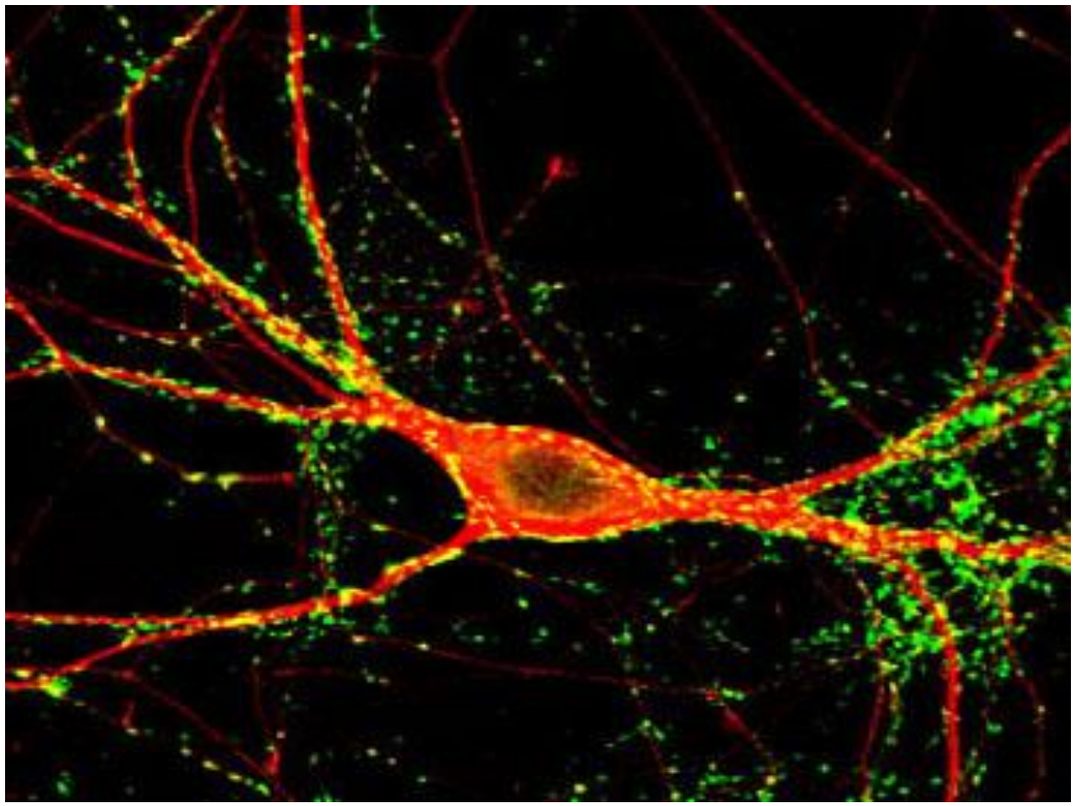


The instantaneous firing rate cannot be computed real-time, due to causality.

Synapses

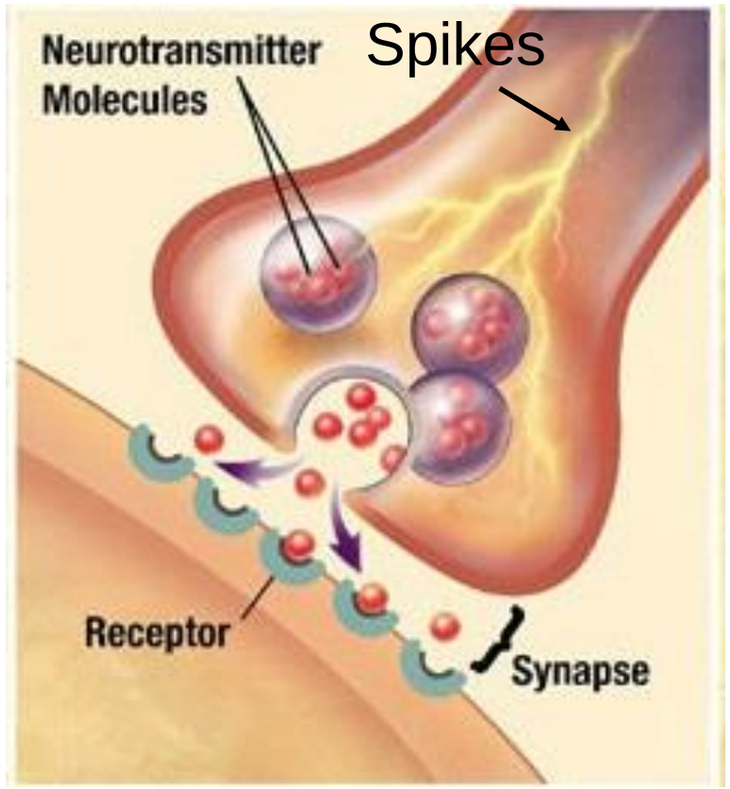
Axons and dendrites are connected through **synapses**. Each neuron has roughly 1000-10000 synapses.



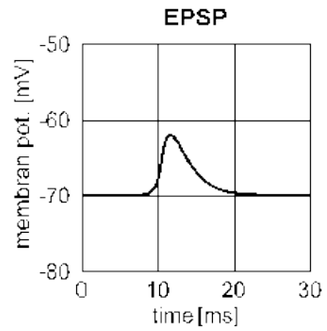


Synapses

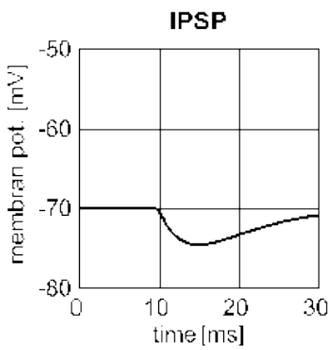
Synapses can chemical or electrical, excitatory or inhibitory:



- a chemical **excitatory** synapse releases Glutamate → opening of ion channels for Na^+ influx → membrane depolarization (membrane potential *increases*);

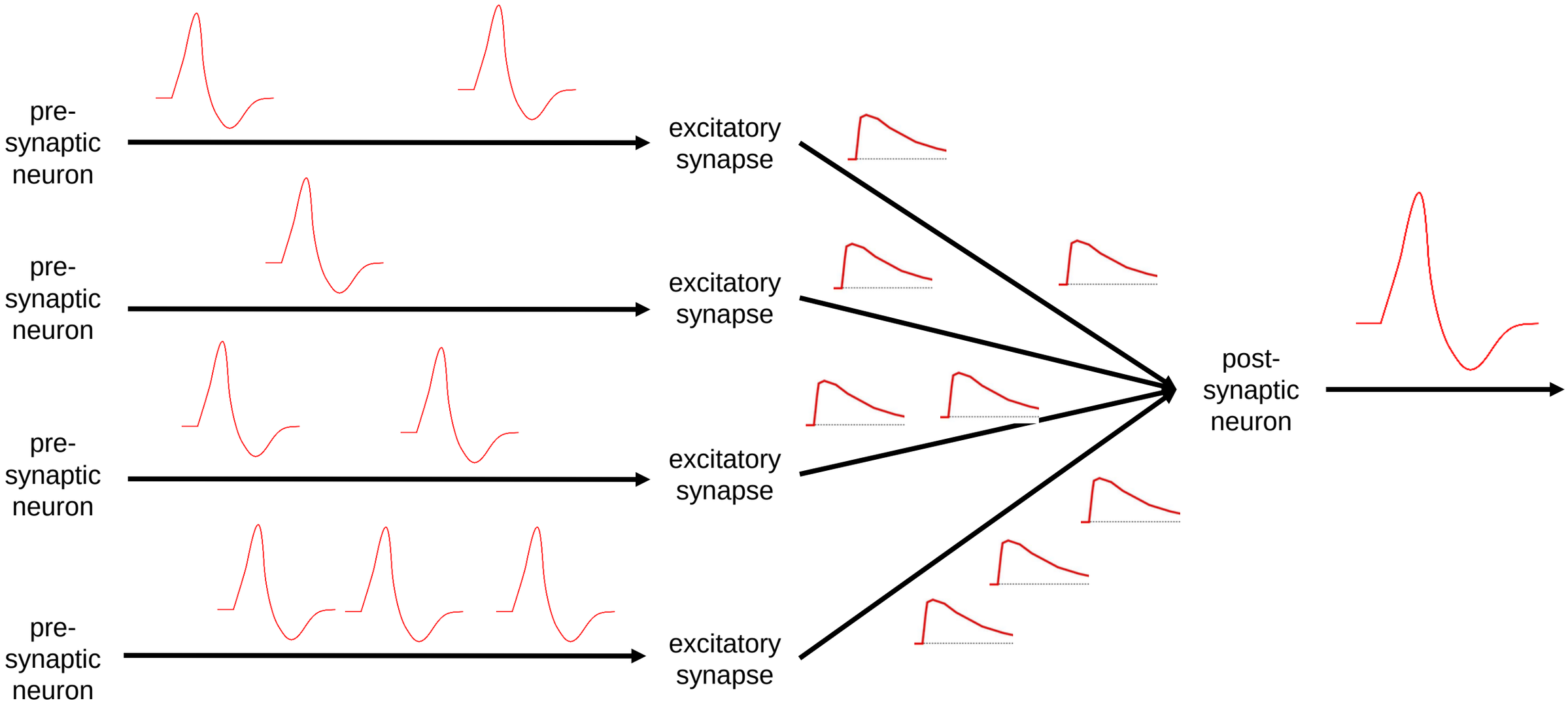


- a chemical **inhibitory** synapse releases GABA neurotransmitter → K+ leaves cell through ion channels → membrane hyperpolarization (membrane potential *decreases*).



Synapses and action potentials

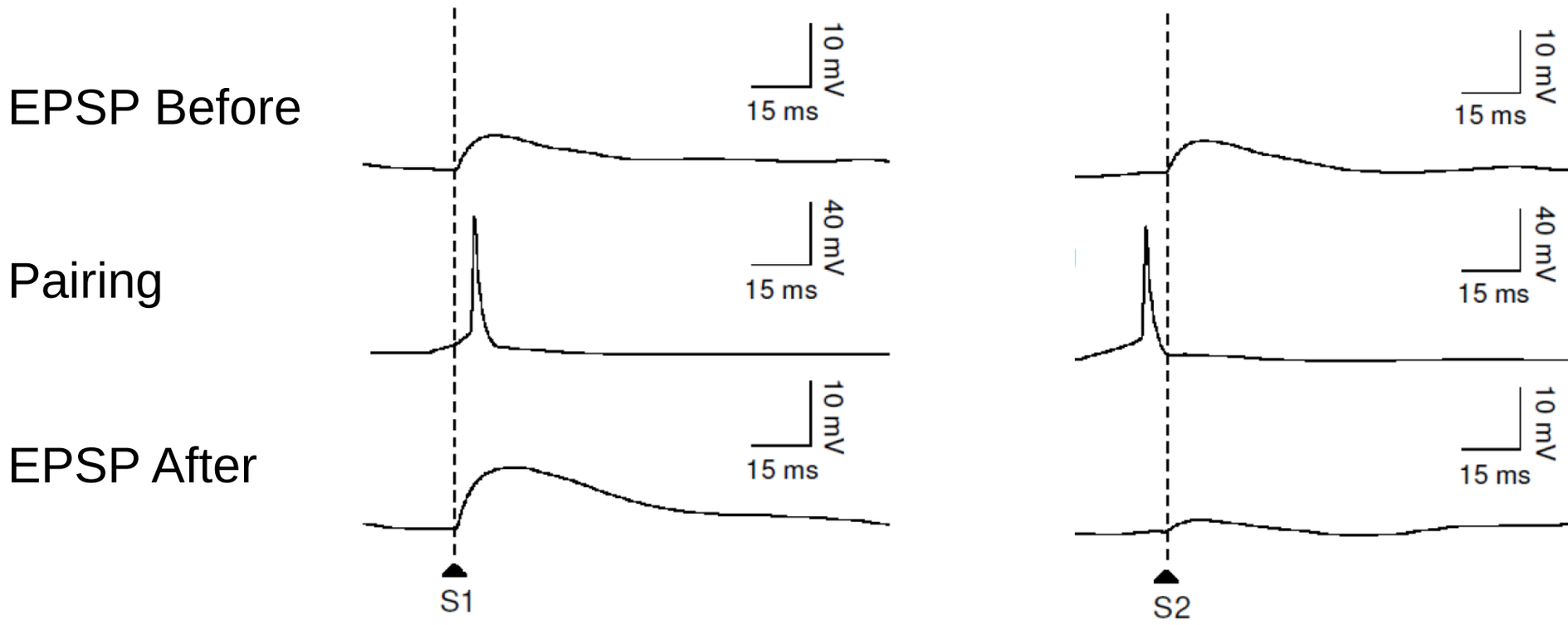
Each spike coming for pre-synaptic neurons and activating excitatory synapses contributes to the generation of an action potential in the post-synaptic neuron.



Synaptic plasticity

Synapses are the basis for memory and **learning**.

If neuron A repeatedly takes part in making neuron B spike, then the synapse from a to B is strengthened and vice versa. This leads to two phenomena:

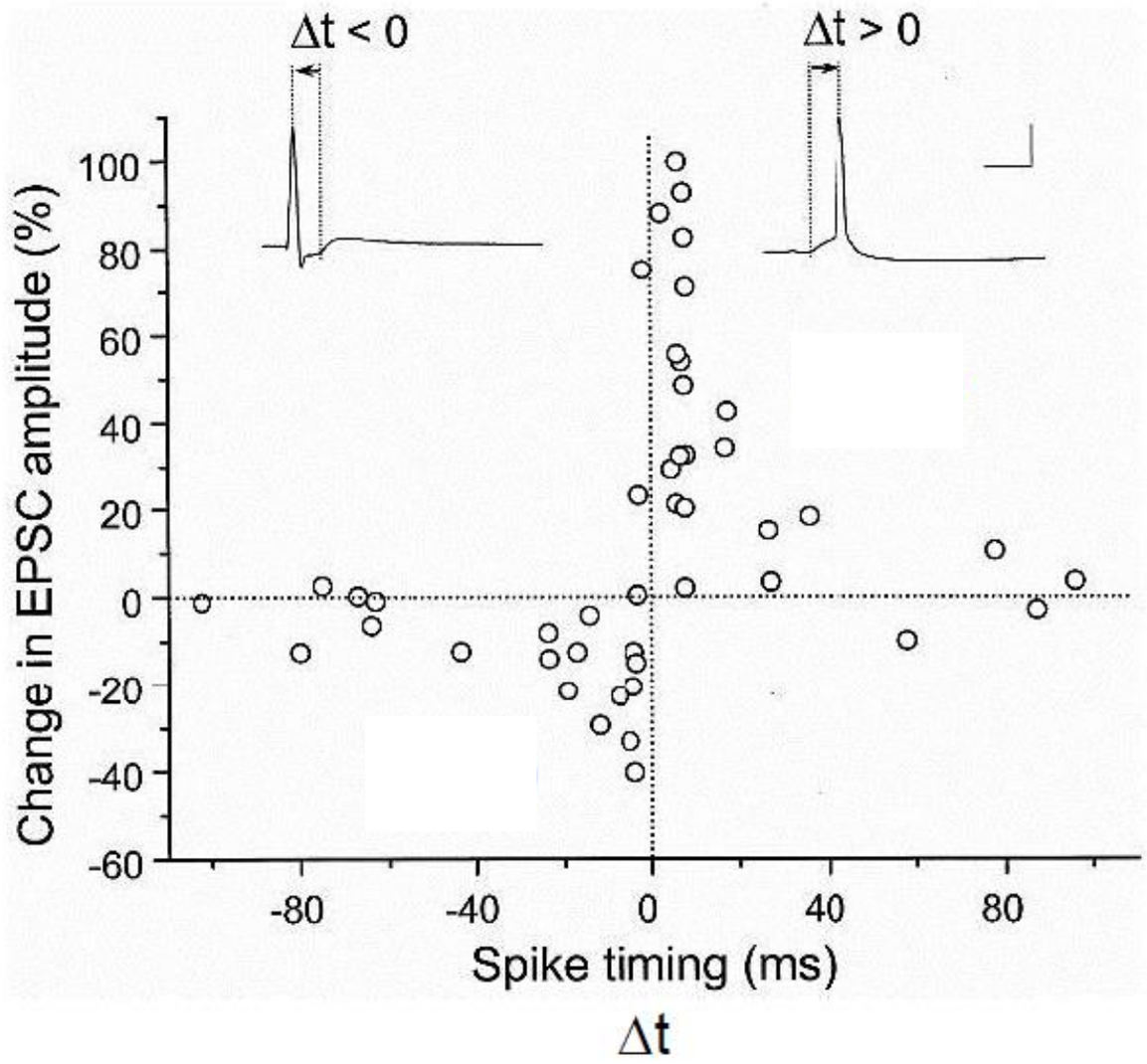


Long term potentiation (LTP)

Long term depression (LTD)

Synaptic plasticity

This adaptation mechanism depends on the timing of the EPSP and the action potential. Thus, it is called **Spike-Timing Dependent Plasticity** (STDP).

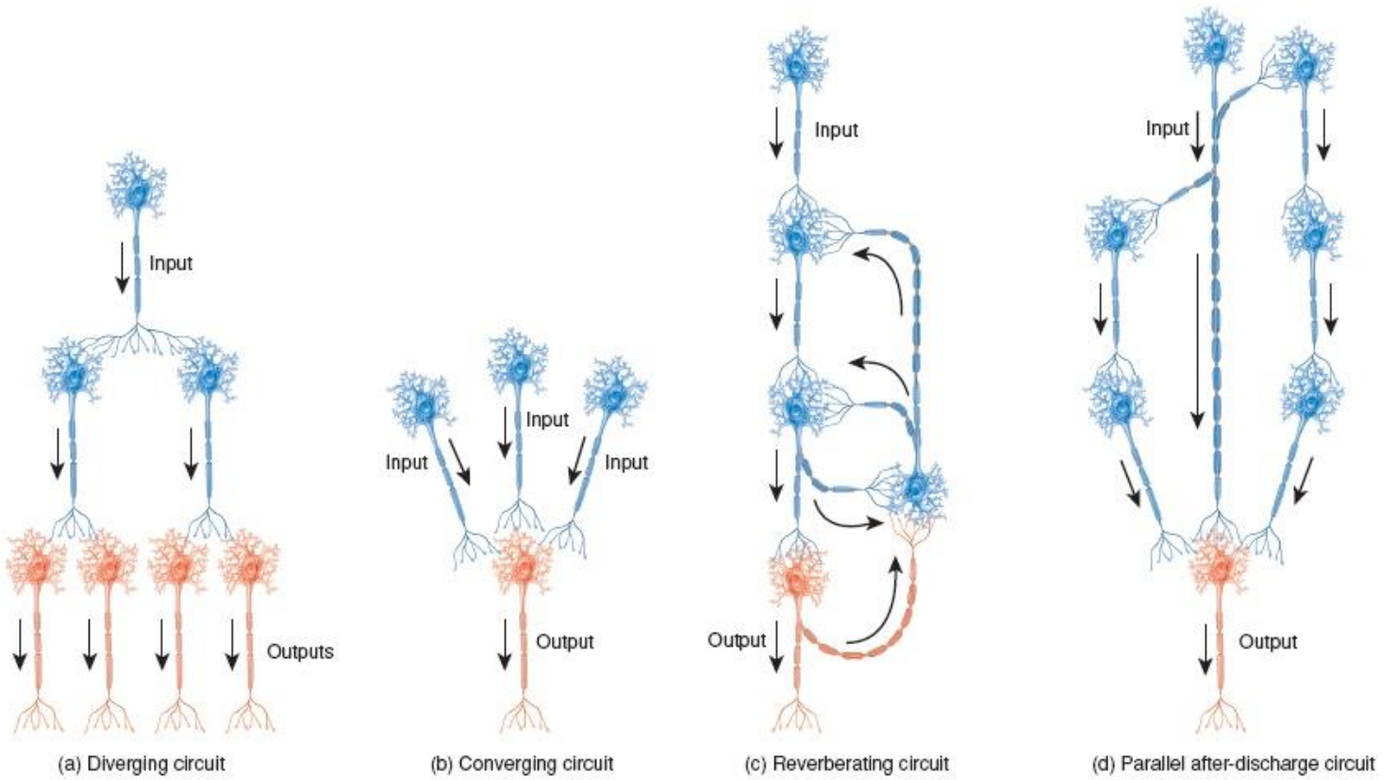


Neural coding

What kind of information can a neuron represent?

None! (by himself)

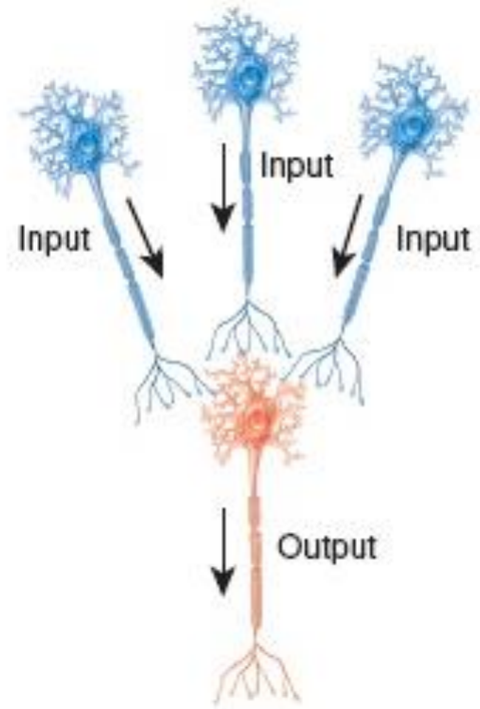
Information is stored in the network topology and synaptic properties.



Receptive fields

A simple way of encoding information is the **receptive field** topology.

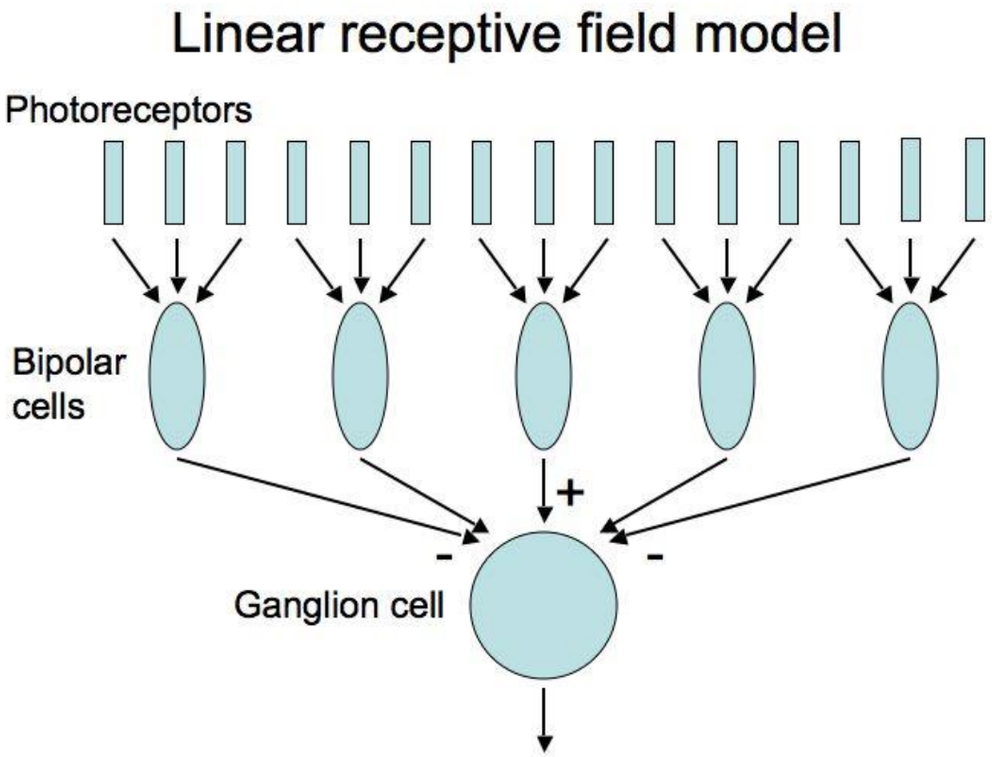
Each receptive field is made up of several input neurons and one output neuron that modulates the combination of their responses.



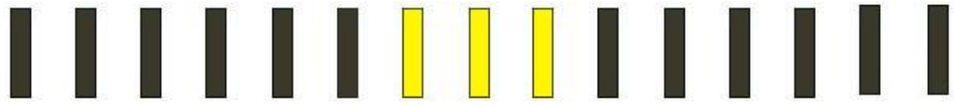
Receptive fields have been identified in the human brain to encode sensory information (auditory system, somatosensory system, visual system).

Receptive fields

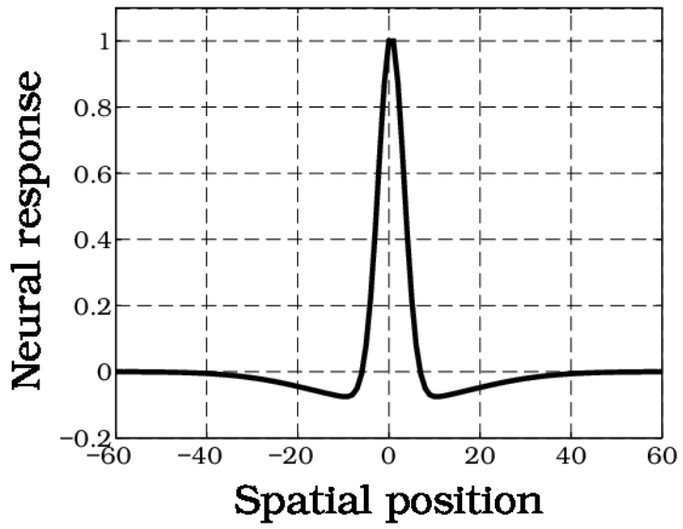
The retinal circuit implements receptive fields to process the image.



Preferred stimulus

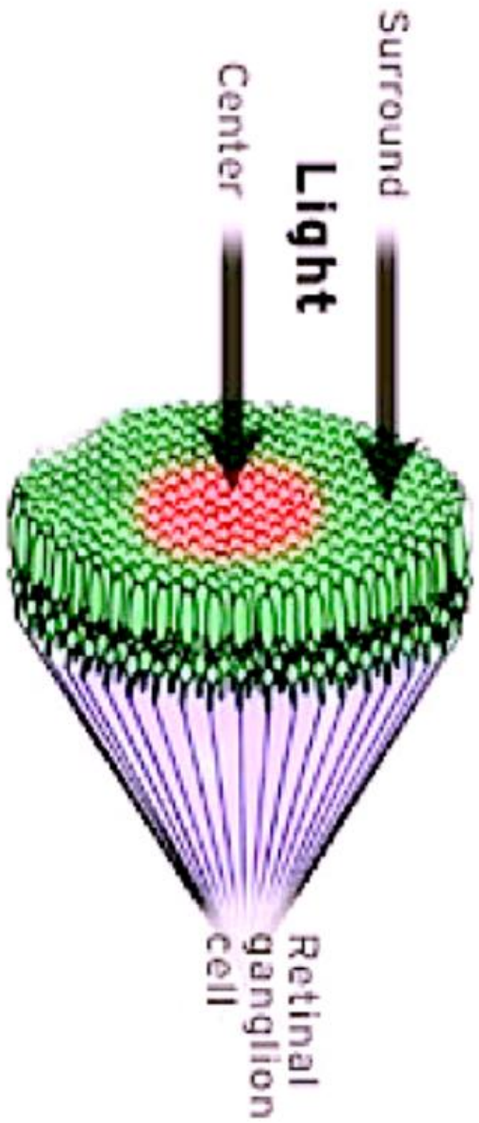


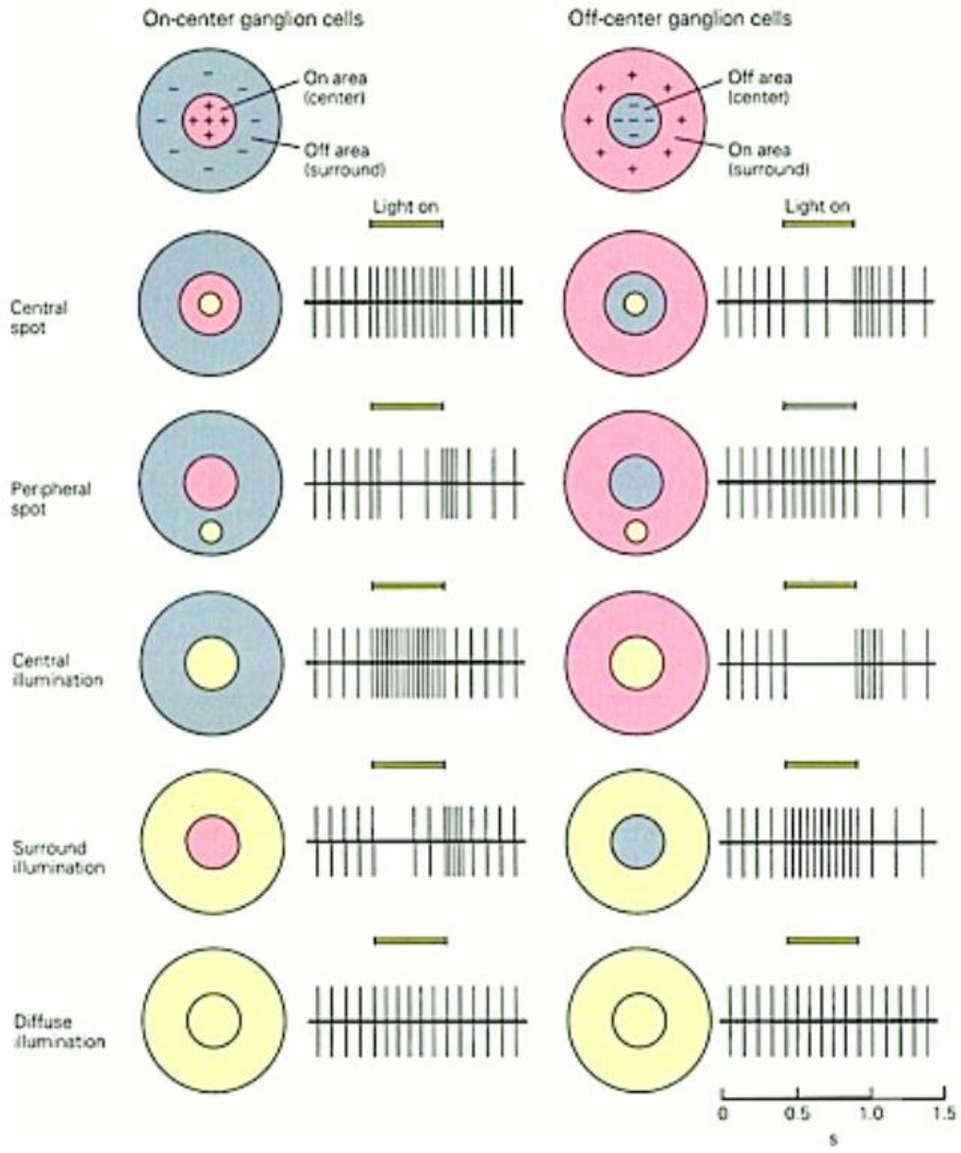
Neural response



Receptive fields

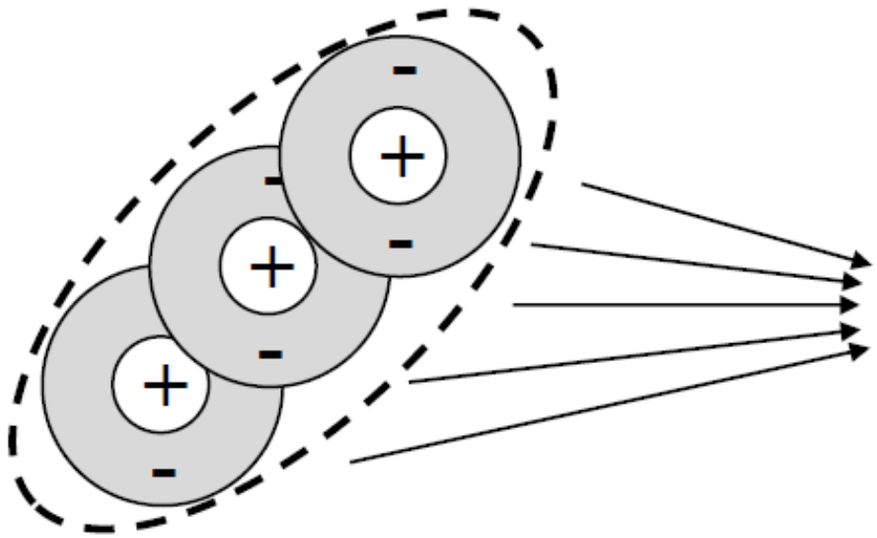
The retinal circuit implements receptive fields to process the image.



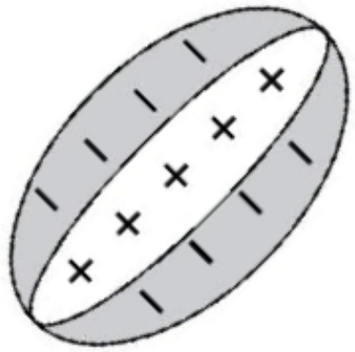


Receptive fields

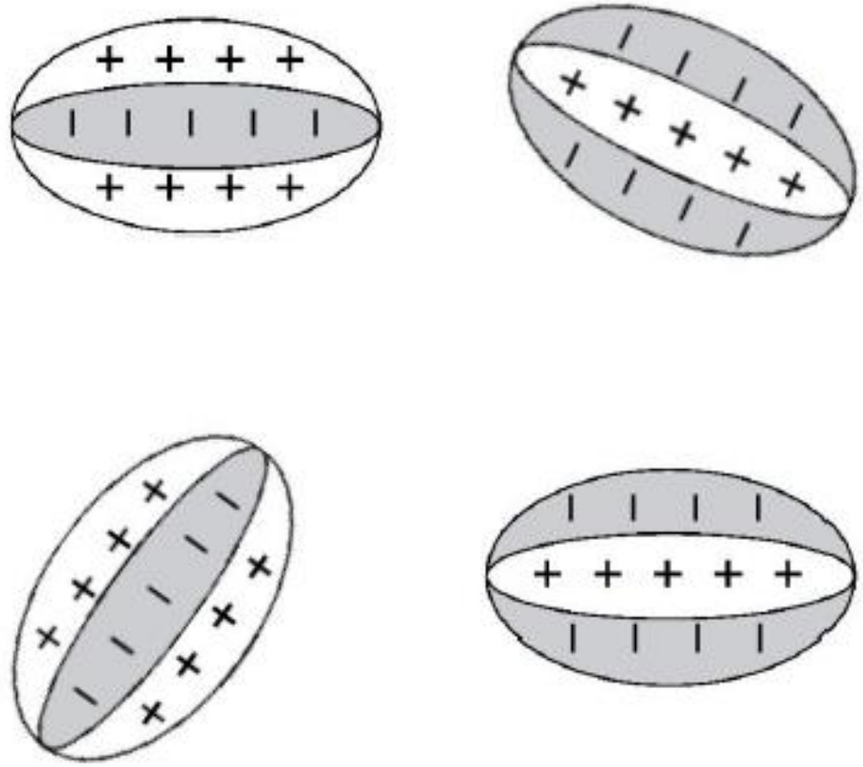
Receptive fields from the retina are in turn used to create oriented receptive fields in the visual cortex.



Retina receptive fields

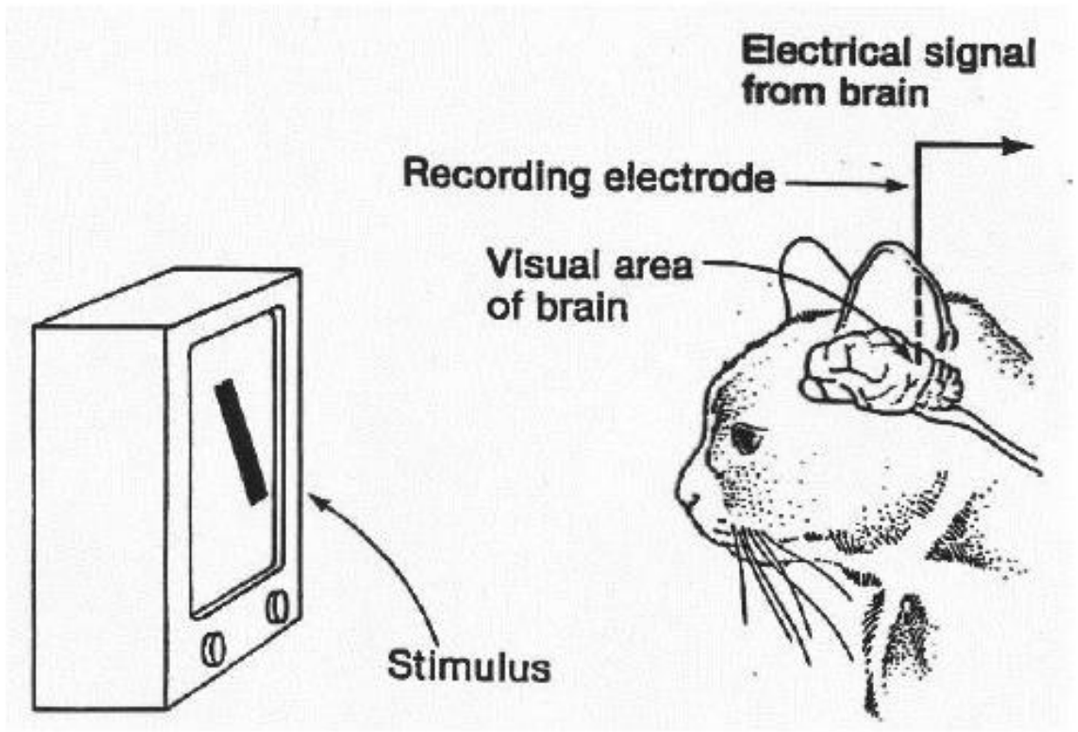


V1 receptive field



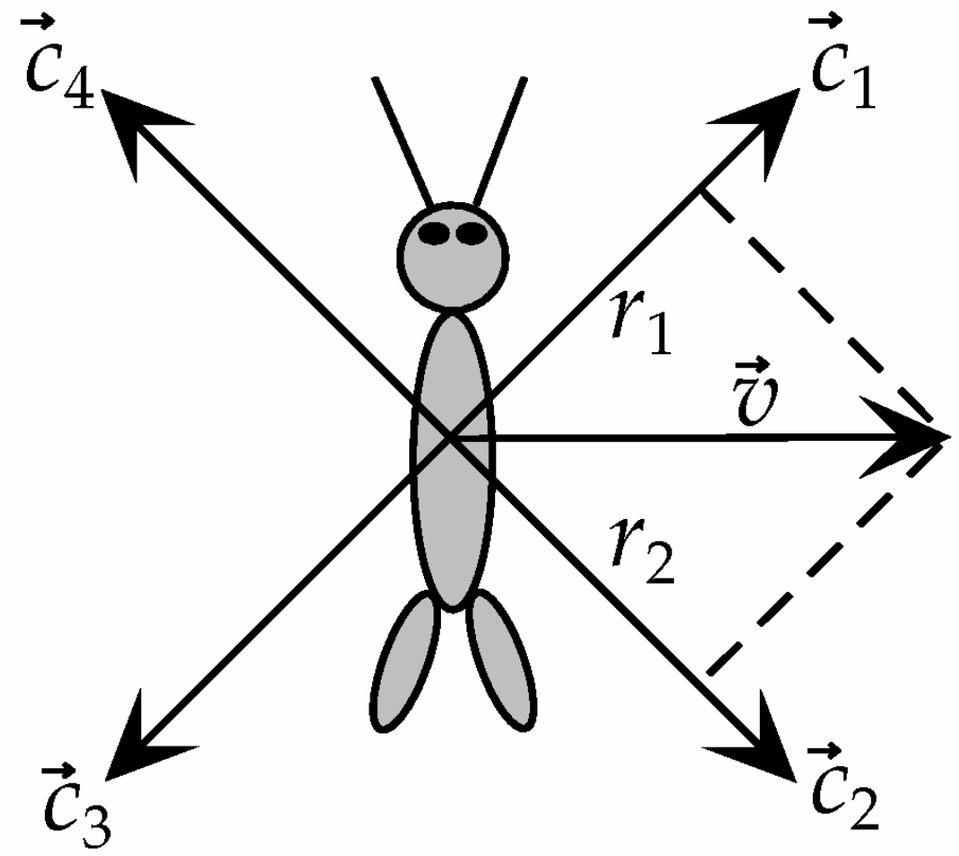
Receptive fields

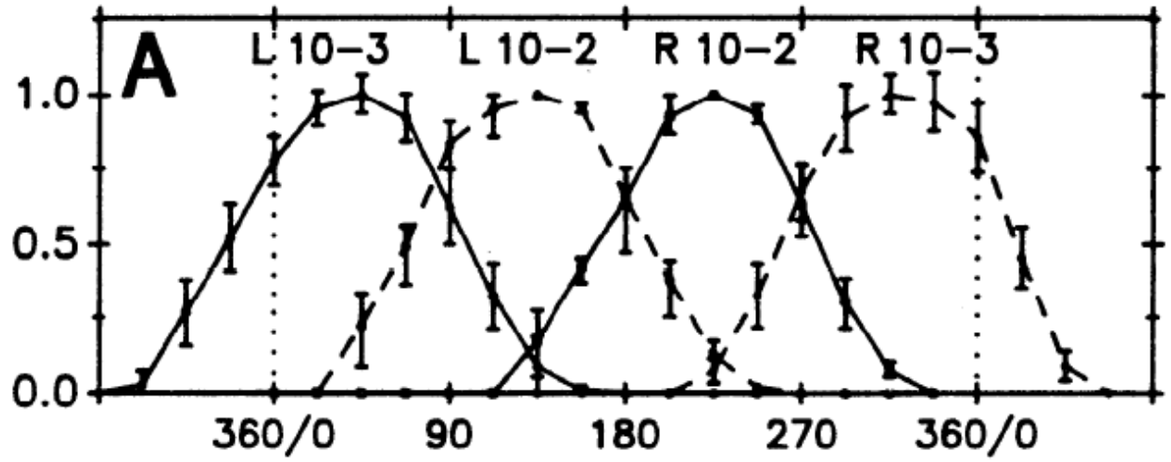
Receptive fields from the retina are in turn used to create oriented receptive fields in the visual cortex (also in animals).



Receptive fields

Each sensory input has its own dedicated brain areas, with receptive fields. This is common feature among animals. Eg. cricket cercal cells.





$$r_i = \vec{v} \cdot \vec{c_i}$$

Is the representation efficient?
Why not using only c_1 and c_2?

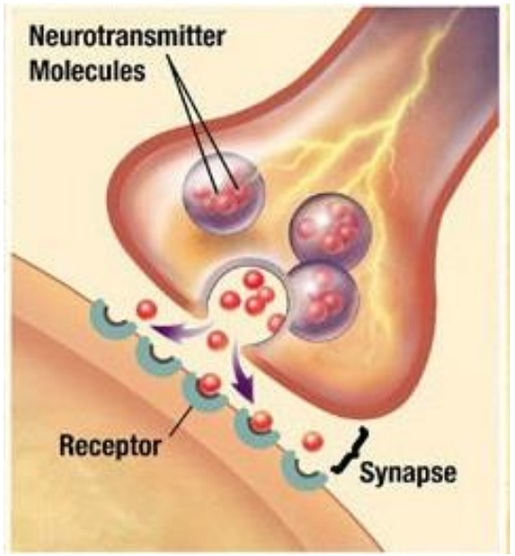
Outline

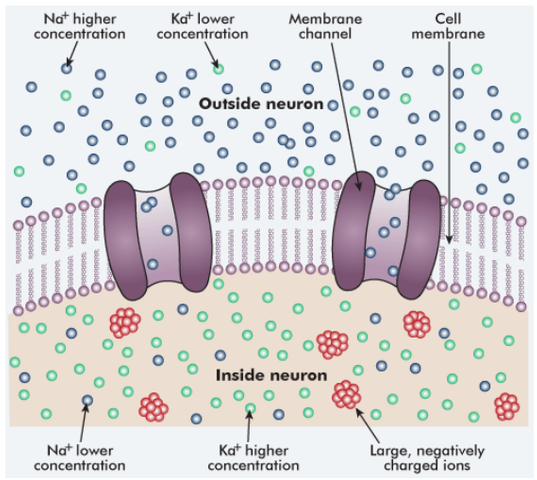
1. Introduction
2. Fundamentals of neuroscience
3. **Simulating the brain**
4. Software and hardware simulations
5. Robotic applications

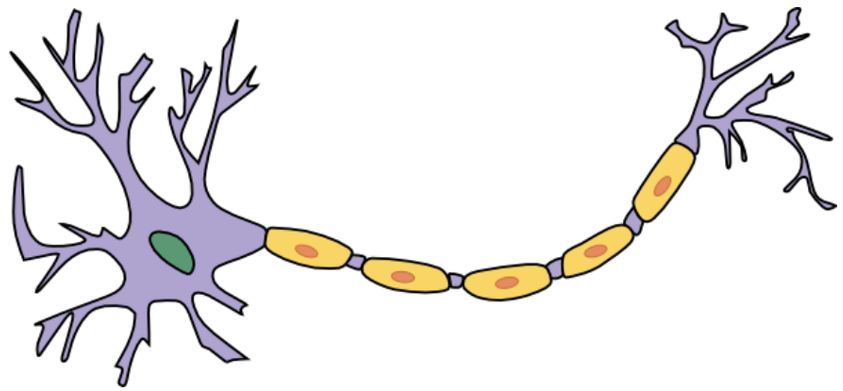
Neuron abstractions

In order to simulate the behaviour of neural circuits we have to model the neuron dynamics.

Thus, we have to translate neurophysiologic properties into equations that we can implement.







Abstract neuron models

- Rate-based
- Point neuron
- Detailed neuron

Detailed neural abstraction

In these kind of models every aspect of the cell morphology is taken into account: diameter of the soma, length of the axons, position of synapses on the dendrites, distribution of ionic channels, neurotransmitter types, etc…

Pros:

- very accurate
- can model any aspect of neural activity

Cons:

- much knowledge is needed to model networks
- simulation times are high

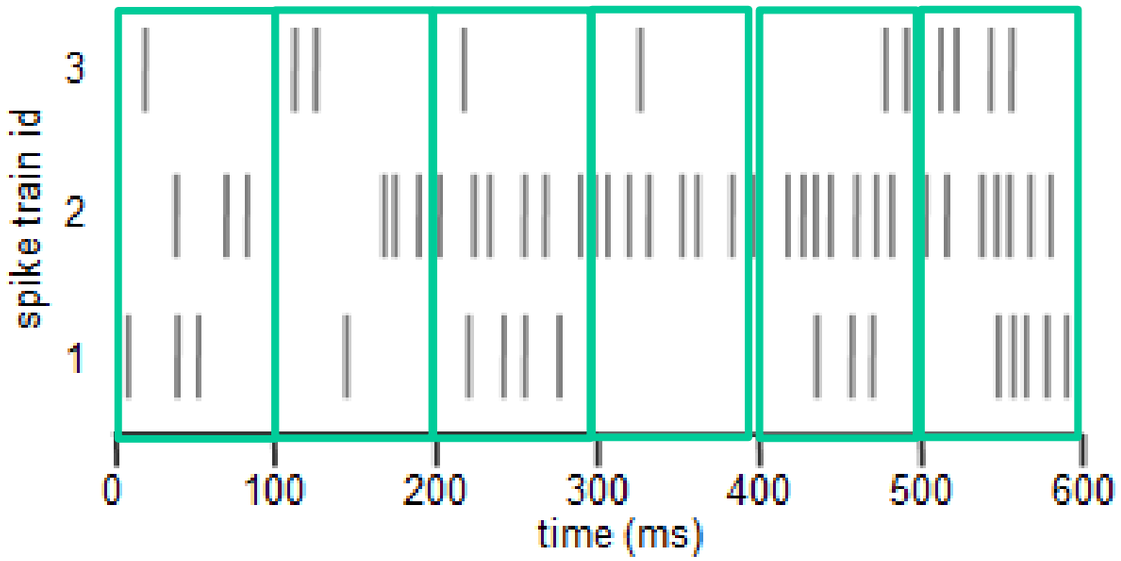
Some detailed neural simulators exist, i.e. NEURON (www.neuron.yale.edu/neuron).

Too little abstraction!

Rate-based abstractions

Each neuron produces spikes with a mean firing rate (in a time interval).

We can sample the firing rate by dividing spikes into bags:



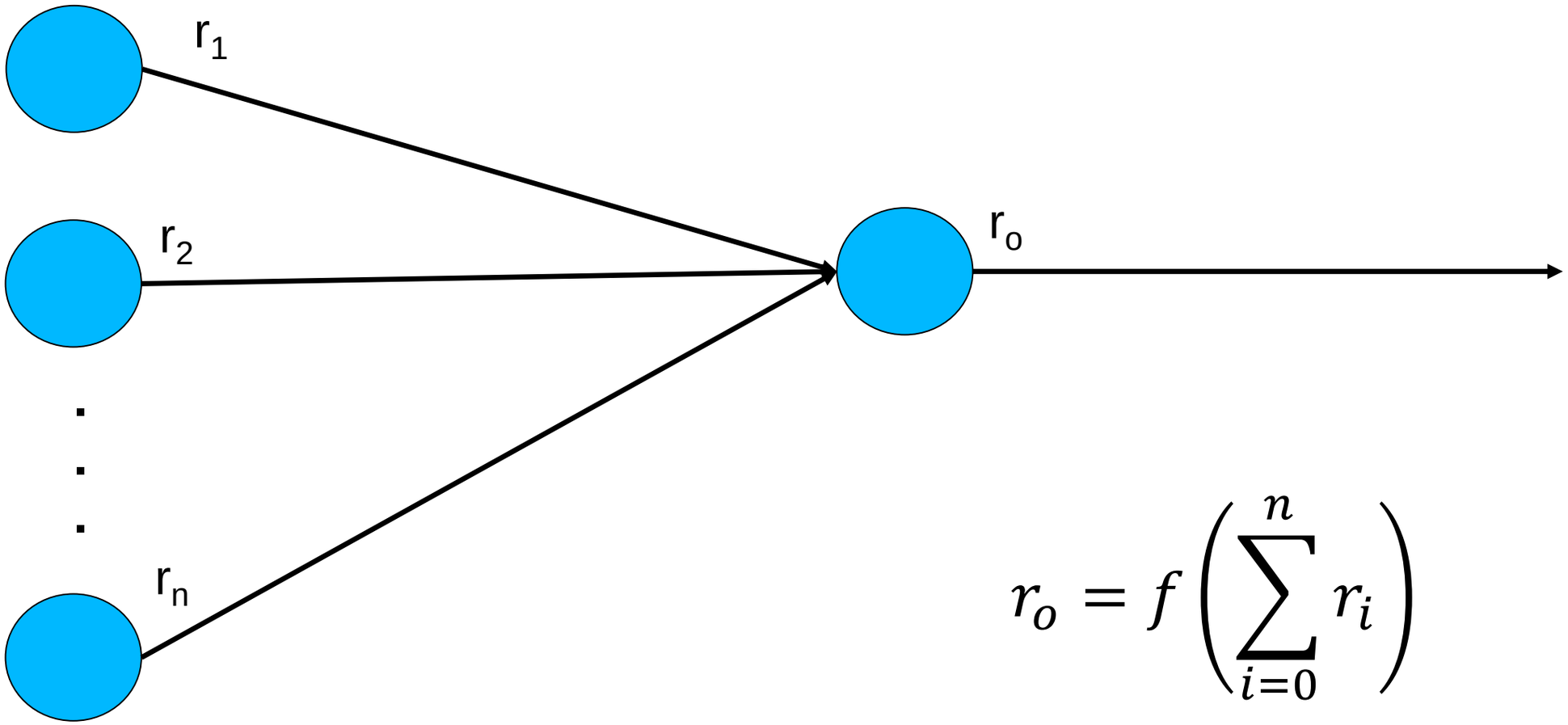
10	20	10	10	20	40
30	30	60	80	70	70
30	10	40	0	30	50

By doing so, we are:

- discretizing time
- forgetting about single action potential events

Rate-based abstractions

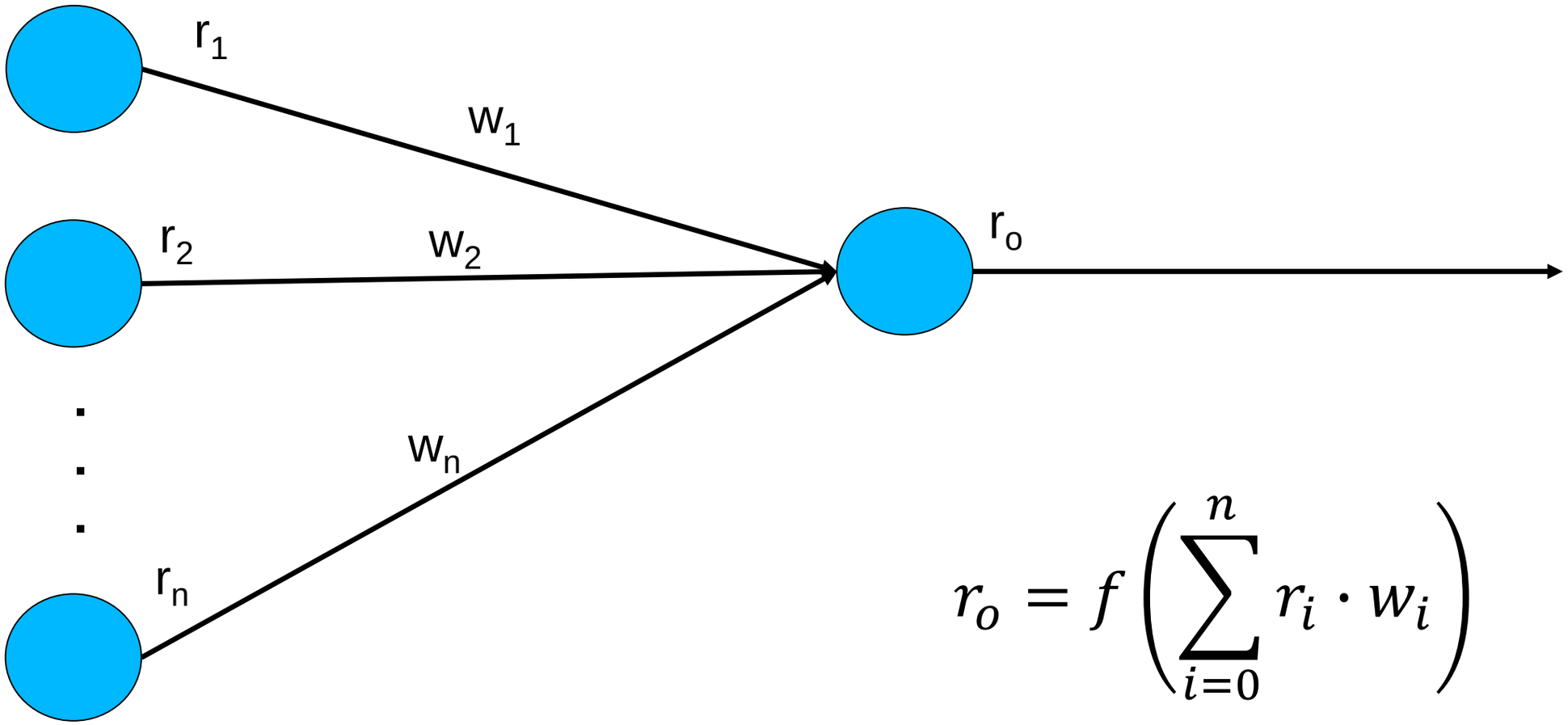
Activity of a post-synaptic neuron can be computed as a function of the rates of pre-synaptic neurons.



$$r_o = f\left(\sum_{i=0}^{n} r_i\right)$$

Rate-based abstractions

What about synapses? We can add weights on the connections.



$$r_o = f\left(\sum_{i=0}^{n} r_i \cdot w_i\right)$$

→ Rosenblatt's perceptron and ANN.

Too much abstraction!

Point neuron abstractions – neuron models

The neuron electrical properties can be described through electrical circuits:

- the lipidic membrane acts as a capacitor (C_m);
- all PSP can be summed up and represented as an external current generator (I_{ext}).

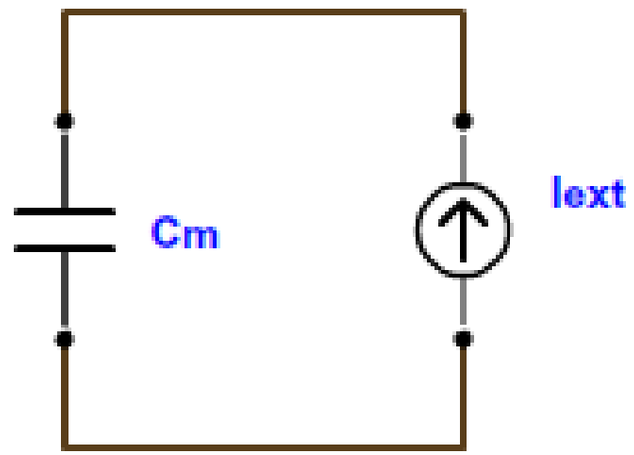
We are interested in the voltage between the two termination of the capacitor (membrane potential, V_m) and we also add the action potential rule:

If $V_m > V_{th}$ then V_m resets to V_{reset} and a spike is emitted.

Point neuron abstractions – neuron models

A first circuit representing neural activity is the **Integrate and fire** model (IAF).

Kirchhoff's law: $$I_C(t) = I_{ext}(t)$$



By deriving the law of capacitance:

$$Q(t) = C_m V_m(t)$$

$$\downarrow$$

$$I_C(t) = C_m \frac{dV_m(t)}{dt}$$

Thus, we obtain:

$$\boxed{\frac{dV_m(t)}{dt} = \frac{I_{ext}(t)}{C_m}}$$

Let's try it out!

Point neuron abstractions – neuron models

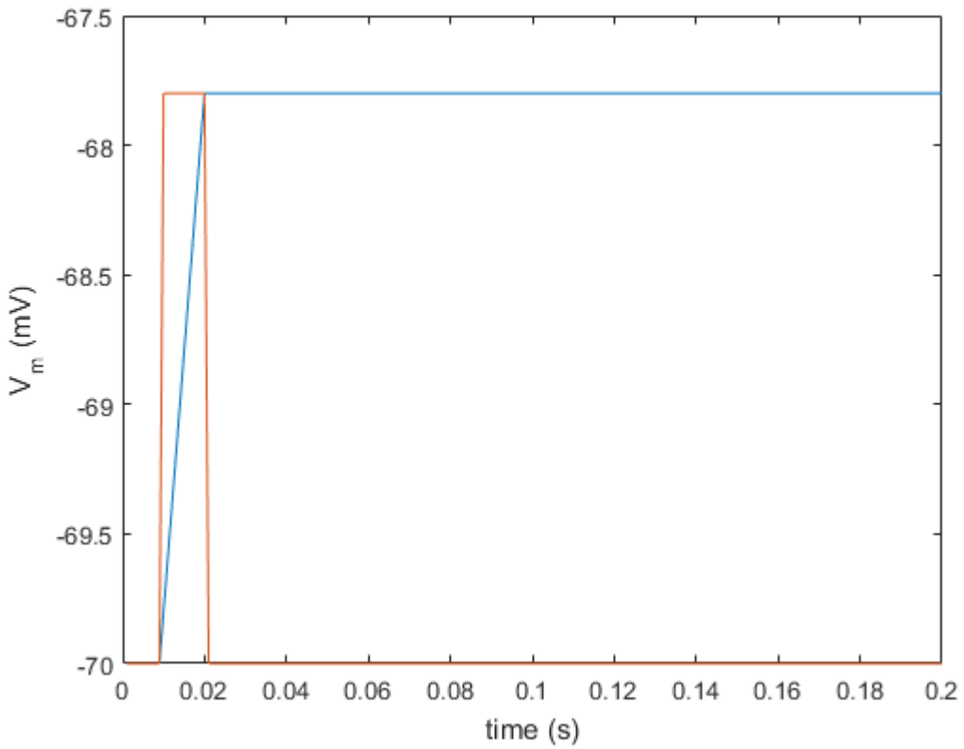
Neurons have the **refractory period**, that must be taken into account for an accurate simulation. Otherwise, the firing rate will rise indefinitely.

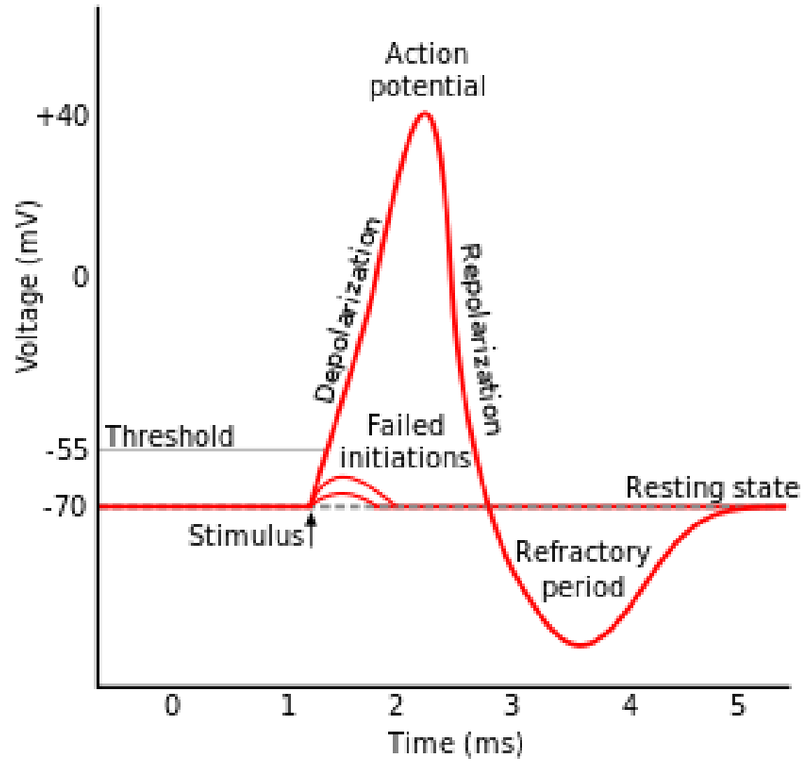
without: $r(I) = \frac{I}{C_m(V_{th} - V_{reset})}$ $\lim_{I \to +\infty} r(I) = +\infty$

with: $r(I) = \frac{I}{C_m(V_{th} - V_{reset}) + t_{ref} I}$ $\lim_{I \to +\infty} r(I) = \frac{1}{t_{ref}}$

Point neuron abstractions – neuron models

In the IAF model, the membrane continues to keep the gained potential, even if there is no external input current and the spike threshold is not reached. This is not true for the biological neuron.

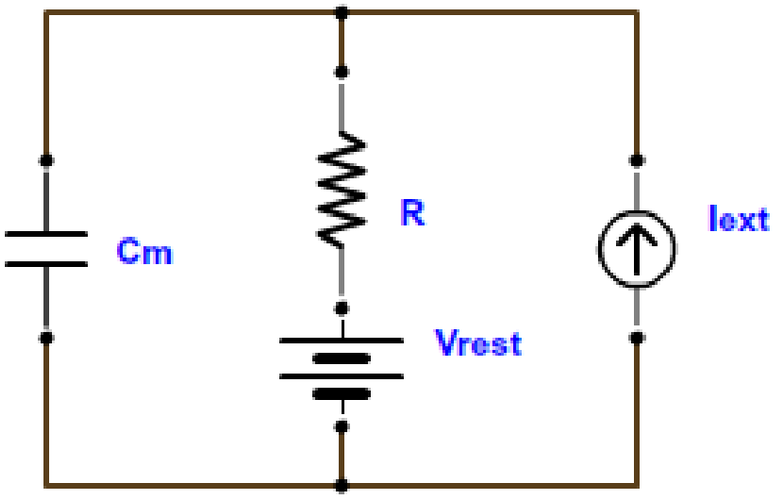




Point neuron abstractions – neuron models

The **Leaky integrate and fire** model (LIAF) adds a resistance in the circuit in order to model the leakage of charge. Moreover, a battery is added to represent the equilibrium potential of the cell membrane.

Kirchhoff's law: $$I_C(t) + I_R(t) = I_{ext}(t)$$



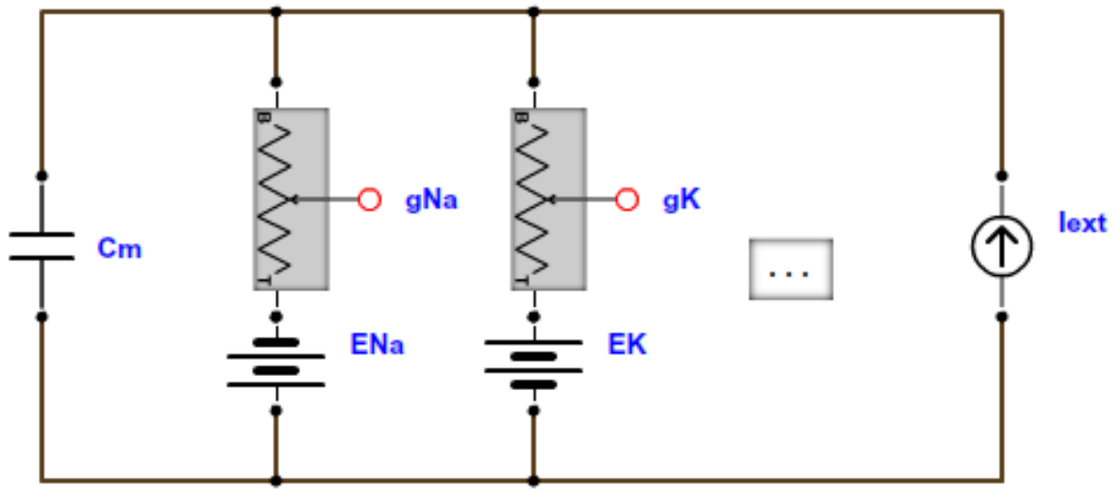
Ohm's law: $$I_R(t) = \frac{(V_m(t) - V_{rest})}{R}$$

Thus, we obtain: $$\boxed{\frac{dV_m(t)}{dt} = \frac{I_{ext}(t)}{C_m} - \frac{(V_m(t) - V_{rest})}{C_m R}}$$

Point neuron abstractions

There are many others neuron models:

Hodgkin–Huxley: each ionic channel is modelled as a resistance-battery parallel circuit, with a probabilistic conductance.



$$\frac{dV_m(t)}{dt} = \frac{I_{ext}(t)}{C_m} - \frac{1}{C_m}\sum_i g_i(V_m(t) - E_i)$$

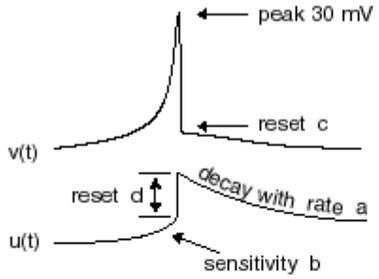
Point neuron abstractions – neuron models

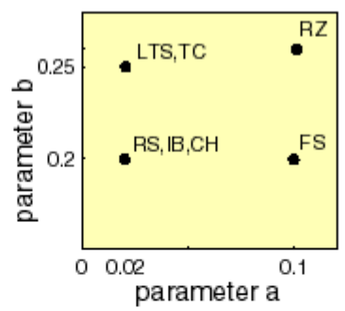
There are many others neuron models:

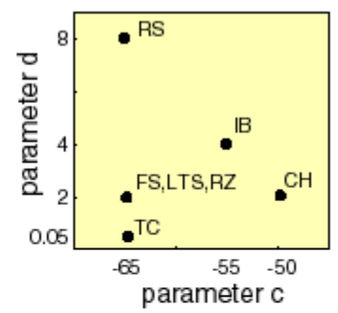
Izhikevich: two differential equations can model many different neuron behaviours.

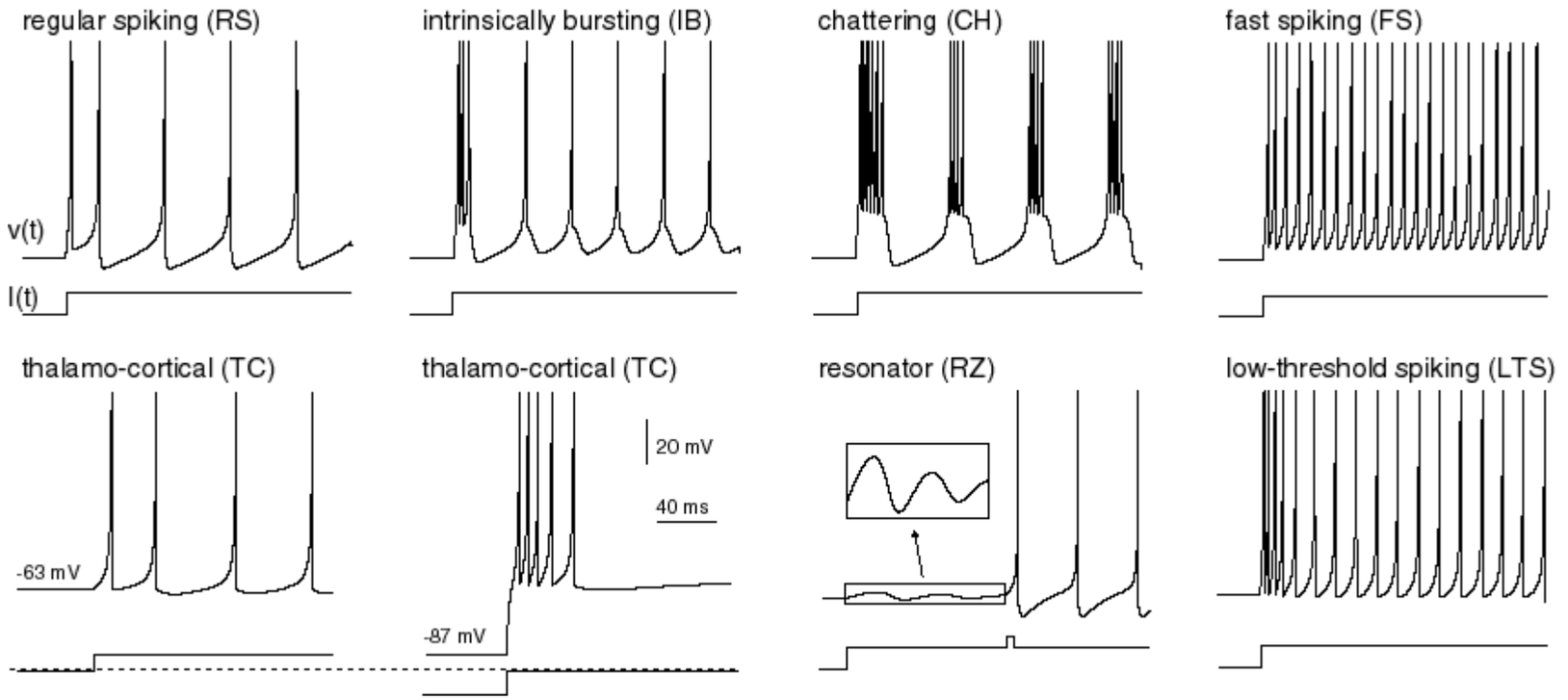
$v' = 0.04v^2 + 5v + 140 - u + I$
$u' = a(bv - u)$

if $v = 30$ mV,
then $v \leftarrow c$, $u \leftarrow u + d$





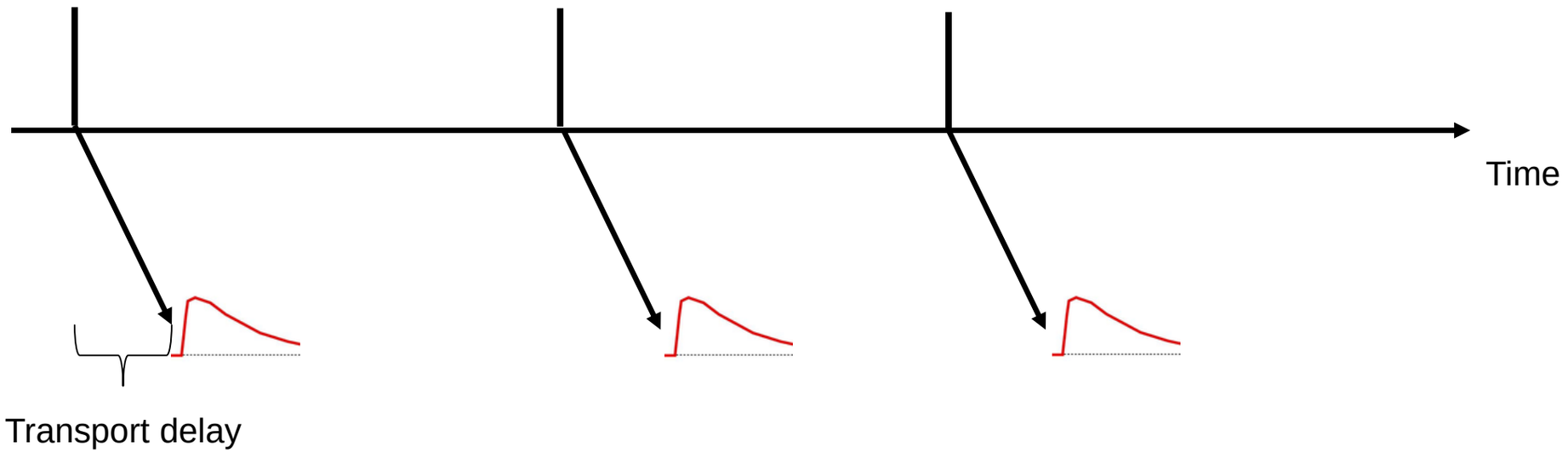




www.izhikevich.com

Point neuron abstractions – synapses models

Each action potential is transmitted as an event to all post-synaptic neurons connected, after a fixed transmission delay. When such event is received a proper EPSP or IPSP is elicited and added to the total input current.



Amongst the most common PSC types there is the **alpha-shaped** one:

$$I(t) = \frac{t}{\tau_s} e^{-\frac{t}{\tau_s}}$$

Point neuron abstractions – synapses models

Each synapse has a weight that has two roles:

1. distinguishing between inhibitory and excitatory synapses by being negative or positive;

2. representing the strength of the connection between the two neurons.

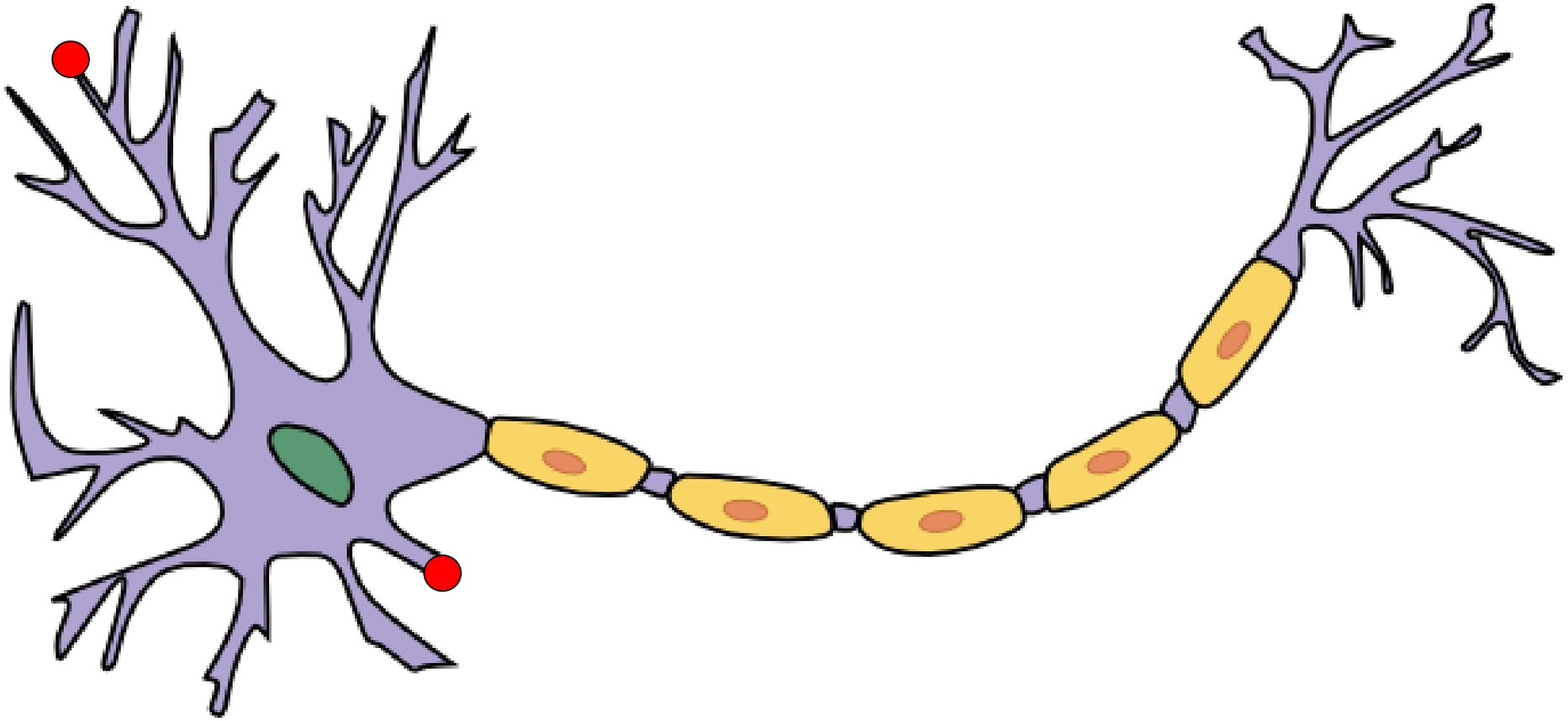
Synaptic weights can be changed via rules implementing STDP, for example:

$$\Delta w_{ij} = \sum_{f}\sum_{n} W(t_i^f - t_j^n) \qquad W(x) = \begin{cases} A_+ e^{\left(-\frac{x}{\tau_+}\right)} for\ x > 0 \\ -A\_e^{\left(-\frac{x}{\tau_-}\right)} for\ x < 0 \end{cases}$$

Point neuron abstractions

Why are these called point-neuron abstractions?

Because we do not take into account the neuron morphology. Each neuron is dimensionless and currents propagate instantaneously from all the receiving synapses.



Outline

1. Introduction
2. Fundamentals of neuroscience
3. Simulating the brain
4. **Software and hardware simulations**
5. Robotic applications

Point neuron simulators - NEST

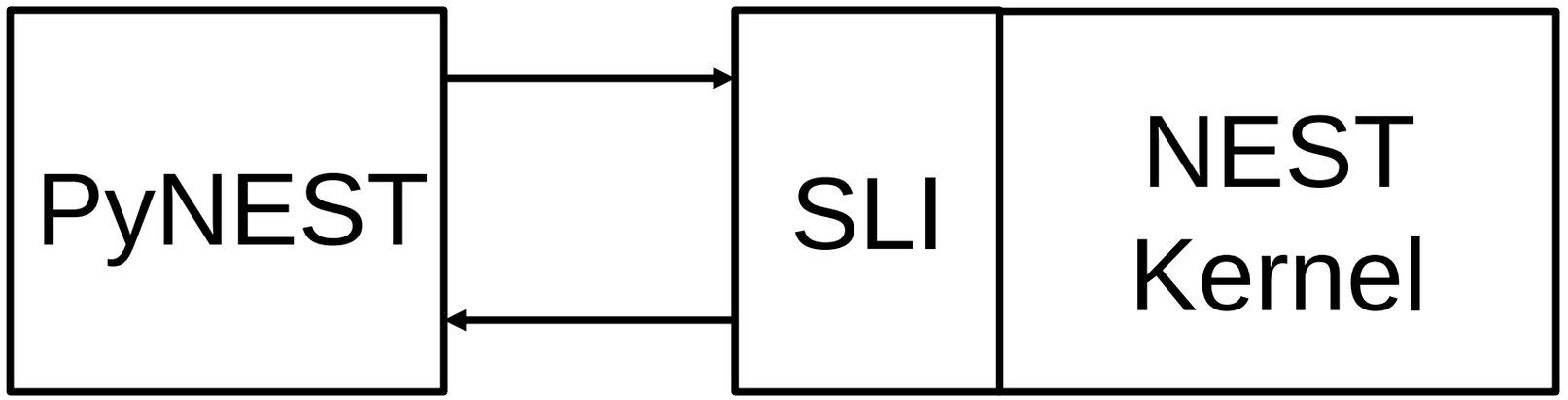
We don’t have to implement a whole simulator by ourselves, several already exists!

Among these, a popular choice is NEST (NEural Simulation Tool), an open source spiking neural network simulator developed by the NEST initiative (www.nest-simulator.org). Among its features, there are:

- over 50 neuron models (including LIAF, Hodgkin-Huxley and Izhikevich)
- over 10 synapse models (including STDP)
- minimal dependencies
- open source (GNU GPLv2)

Point neuron simulators - NEST

NEST has a simulation kernel (written in C++) and two layers of interface towards it.



The kernel cannot be directly accessed. In fact, the executable launches the Simulation Language Interpreter to which one can send commands to create the network.

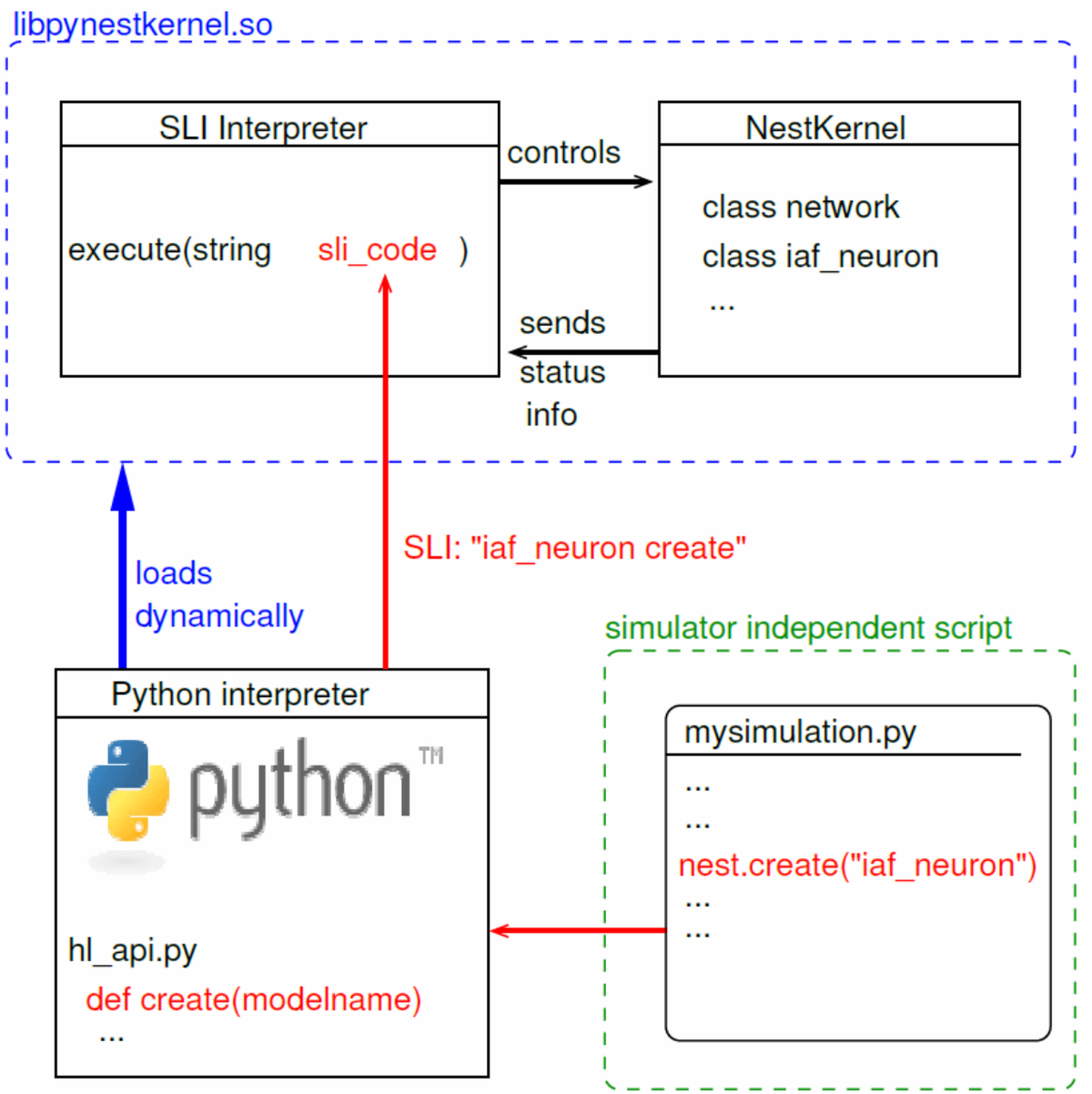
Why PyNEST? Because SLI is basically PostScript!

```
/iaf_neuron Create /n Set
/poisson_generator Create /pg Set
pg << /rate 220.0 Hz >> SetStatus
pg n Connect
```

```
n = nest.Create('iaf_neuron')
pg = nest.Create('poisson_generator')
nest.SetStatus(pg, {'rate': 220.0})
nest.Connect(pg, n)
```

Point neuron simulators - NEST

PyNEST provides an usable interface towards SLI.



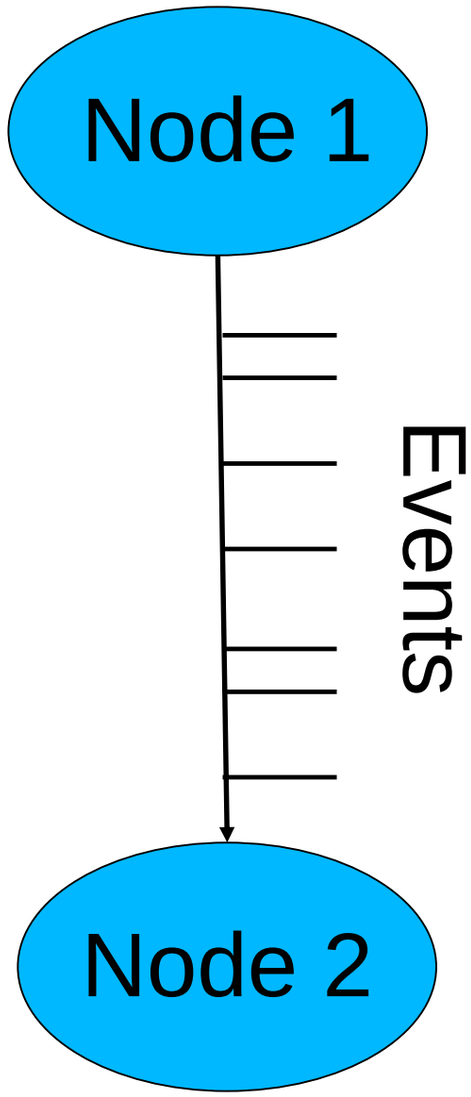
Point neuron simulators - NEST

A NEST network is a *directed weighted graph*:

- **Nodes**
  - neurons, devices, sub-networks
  - have a dynamic state that changes over time and can be influenced by events

- **Events**
  - pieces of information of a particular type (e.g. spike, voltage or current event)

- **Connections**
  - communication channels for the exchange of events
  - directed (pre to post)
  - weighted (synaptic weights)
  - delayed (delay must be greater than 0!)



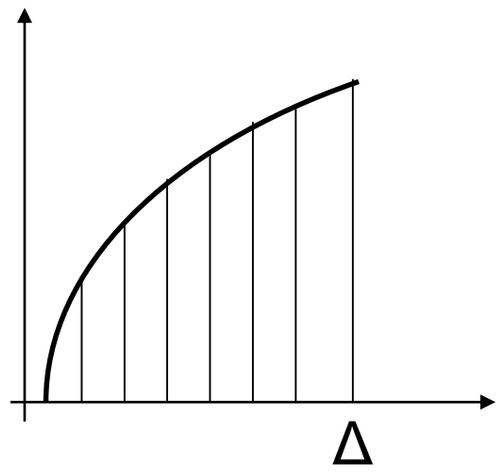
Point neuron simulators - NEST

The simulation is discretized into time steps of a certain duration (Δ). The simulation loop works as follows:

1. PSC for all delivered events are computed
2. membrane potential is updated and new events are bufferized
3. new events are sent towards post-synaptic nodes
4. simulation time is increased by Δ

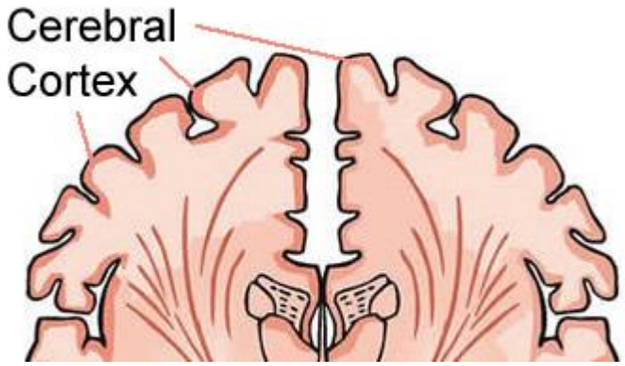
Notes:

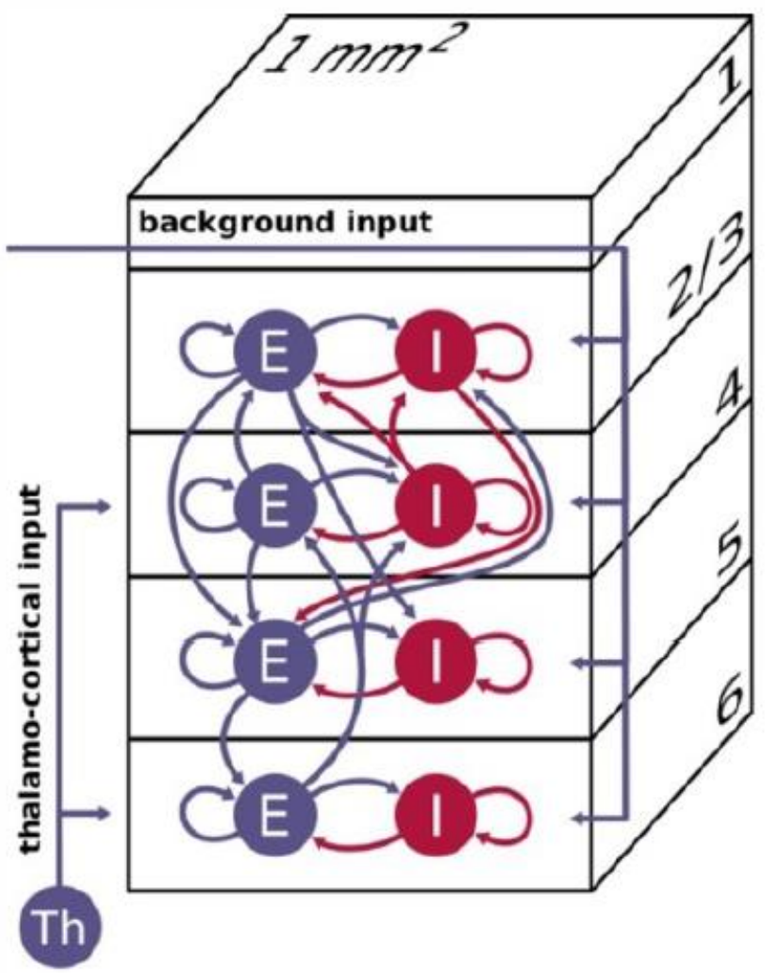
- actually 1 and 2 occur inside an inner loop with a time step < Δ!
- delay of connections must be >= Δ



NEST example – cortical microcircuit

We want to simulate a layer of the cerebral cortex:





- 1mm2
- 0.3 billion synapses, 80000 neurons
- 6 layers
- 2 population of LIAF neurons per layer

Potjans, Tobias C., and Diesmann, Markus. “The cell-type specific cortical microcircuit: relating structure and activity in a full-scale spiking network model.” Cerebral Cortex 24.3 (2014): 785-806

Let’s try it out!

NEST example – cortical microcircuit

What did we learn from this example?

- using nest is very easy to set up neural simulation
- nice syntactic sugar for randomized connection and weights
- useful spike recording utilities
- but it took more than 10 seconds to simulate 1!

```
start_time = time.time()
nest.Simulate(T)
elapsed_time = time.time() - start_time
print elapsed_time
```

```
11.4397270679
```

We need to find a way to *speed up* the simulation.

NEST – parallel simulations

Let’s recall the kernel simulation loop:

1. PSP for all delivered events are computed
2. membrane potential is updated and new events are bufferized
3. new events are sent towards post-synaptic nodes
4. simulation time is increased by Δ

MPI thread

MPI process

Inside a single time step, each neuron is decoupled from the others, thus the simulation of a single time step is an embarrassingly parallel problem.

In fact, NEST natively supports MPI and the parallelization of the loop.

Moreover, MPI is supported on High Performance Computing platforms!

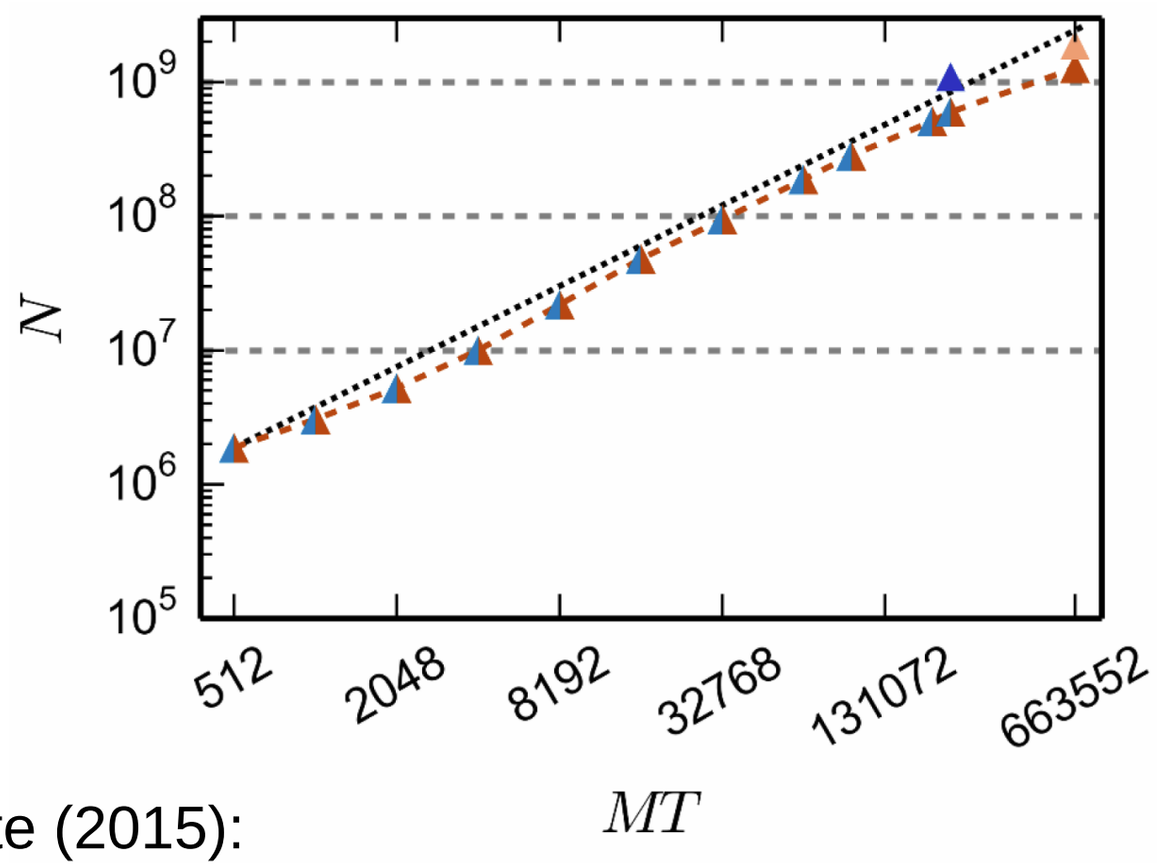
NEST – HPC

How well does nest perform on supercomputers?

Legend:

K - RIKEN, Japan
663,552 nodes, 4th

JUQUEEN - Jülich, Germany
229,376, 11th



Largest network simulation performed to date (2015):

1.86x10^9 neurons, 6000 synapses each

1.08x10^9 neurons, 6000 synapses each

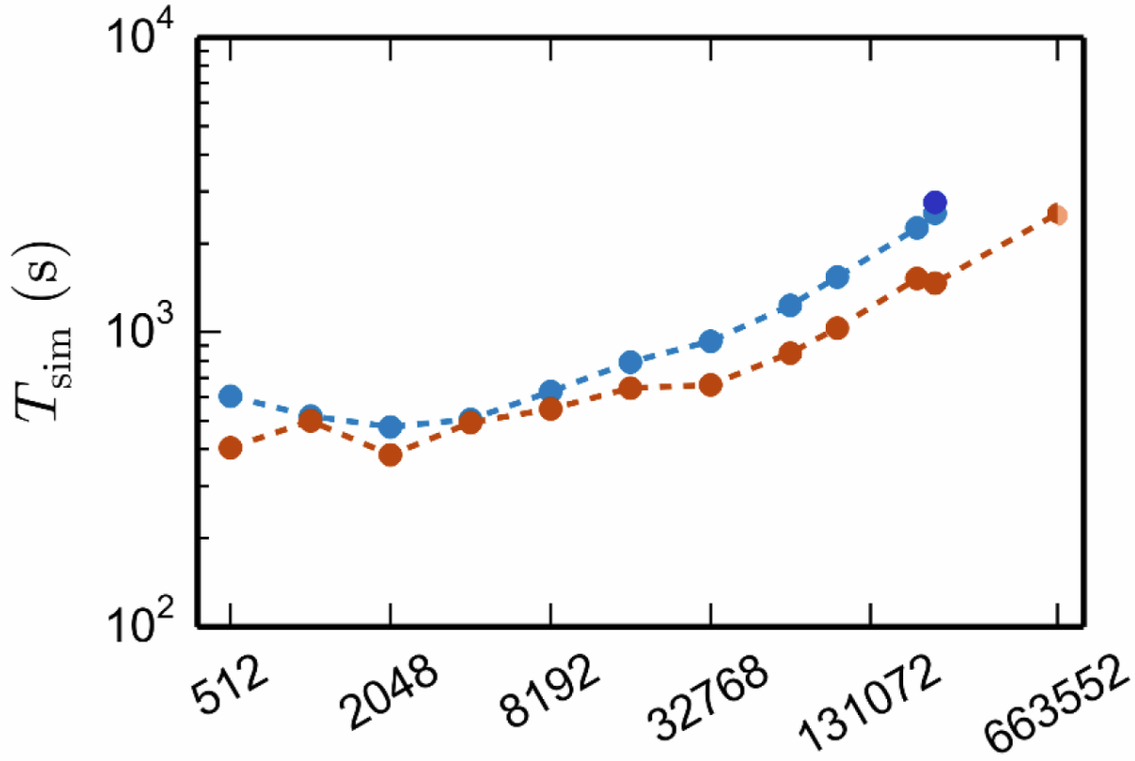
NEST – HPC

How well does nest perform on supercomputers?

Legend:

K - RIKEN, Japan
663,552 nodes, 4th

JUQUEEN - Jülich, Germany
229,376, 11th



Simulation time of 1 second of real time varies between:

between 6 and 42 minutes

between 8 and 41 minutes

NEST – HPC

Is this a viable solution for robotics? Not really.

- even if we would like to simulate smaller networks, simulations will not be real-time
- usually robotics labs do not have supercomputers
- supercomputers work with job systems and as of today no interactive job mechanism exists
- latencies between the supercomputer and the robot
- power consumption
- last but not least, supercomputers do not support Python

Neuromorphic hardware

A new kind of processors, specifically designed to compute neural dynamics, have been developed in the last few years. This is what is called **neuromorphic hardware**.

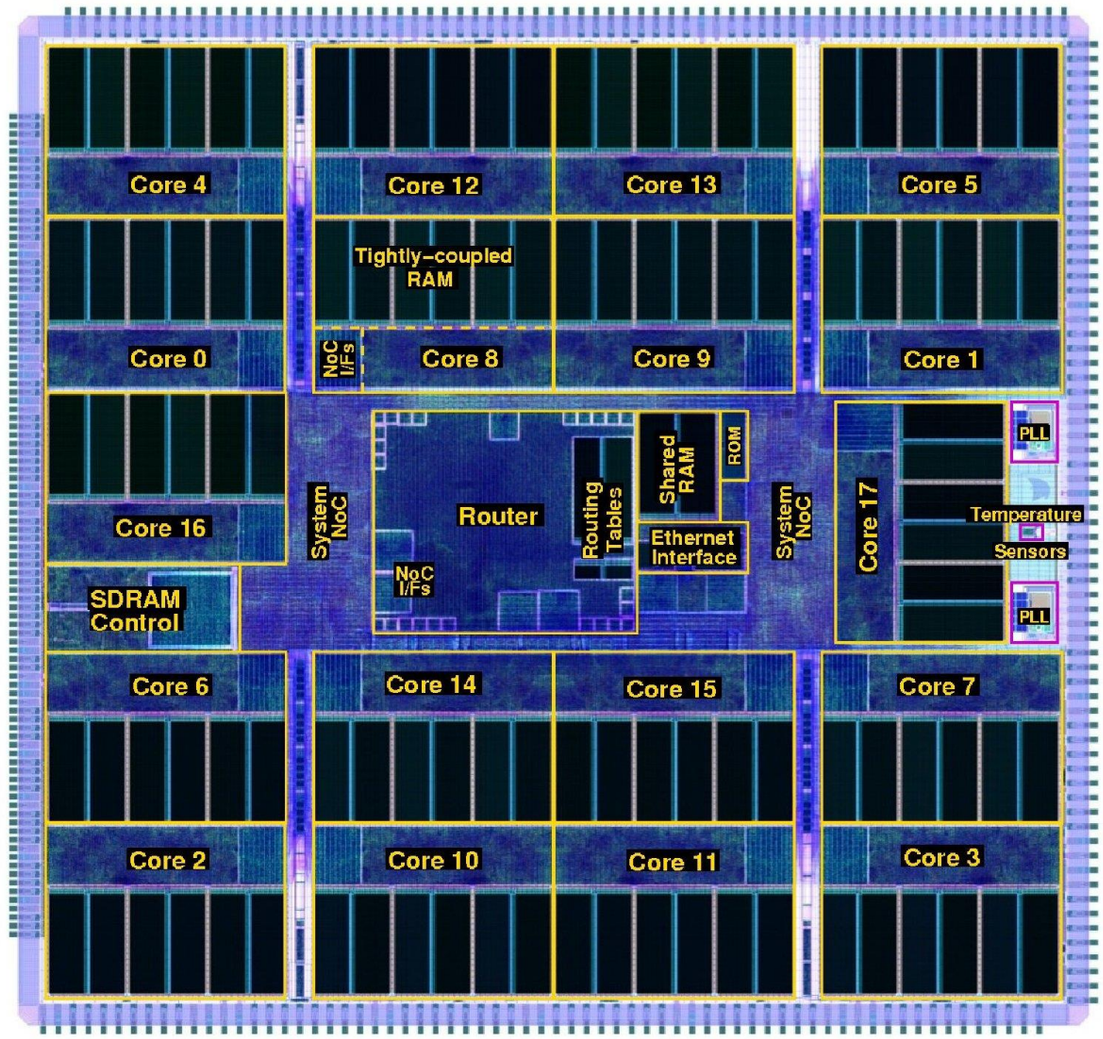
Usually, these kind of processors have these characteristics:

- massively parallel computation
- energy efficiency
- fault tolerance
- self organization of the network
- fast simulation times
- compactness

Neuromorphic hardware – SpiNNaker

SpiNNaker is a neuromorphic hardware platform developed by the University of Manchester.

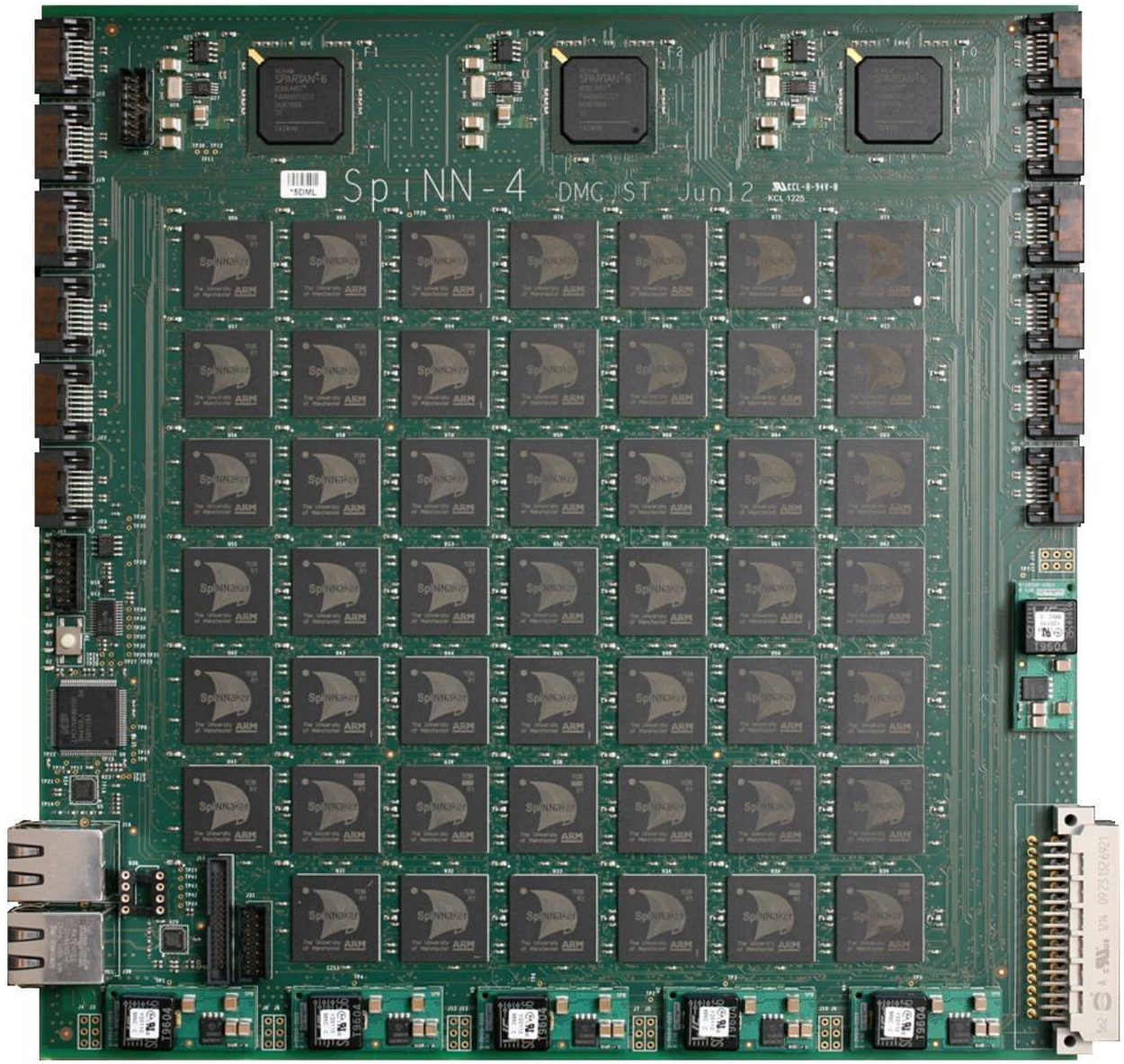
- 1W chip
- 18 ARM-968 cores
- 1Gbit DDR-2 SDRAM
- 240MHz
- 6 bi-directional links
- optimized for 10million 32-bit packets/s



Neuromorphic hardware – SpiNNaker

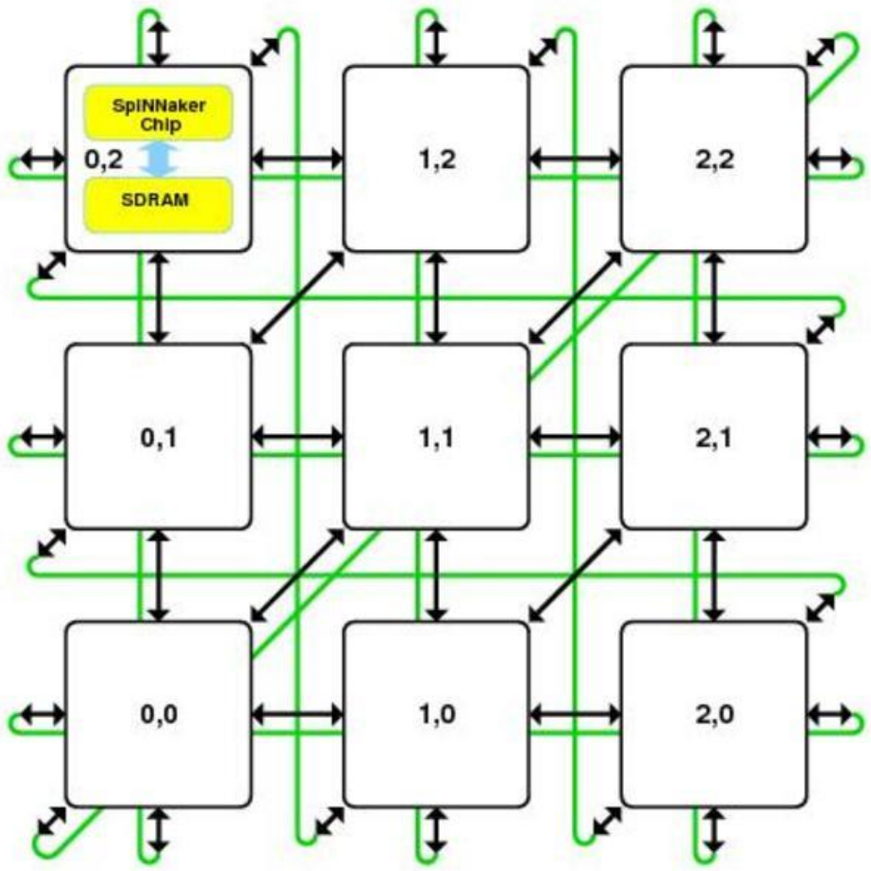
SpiNNaker cores are arranged on 48 chips boards.

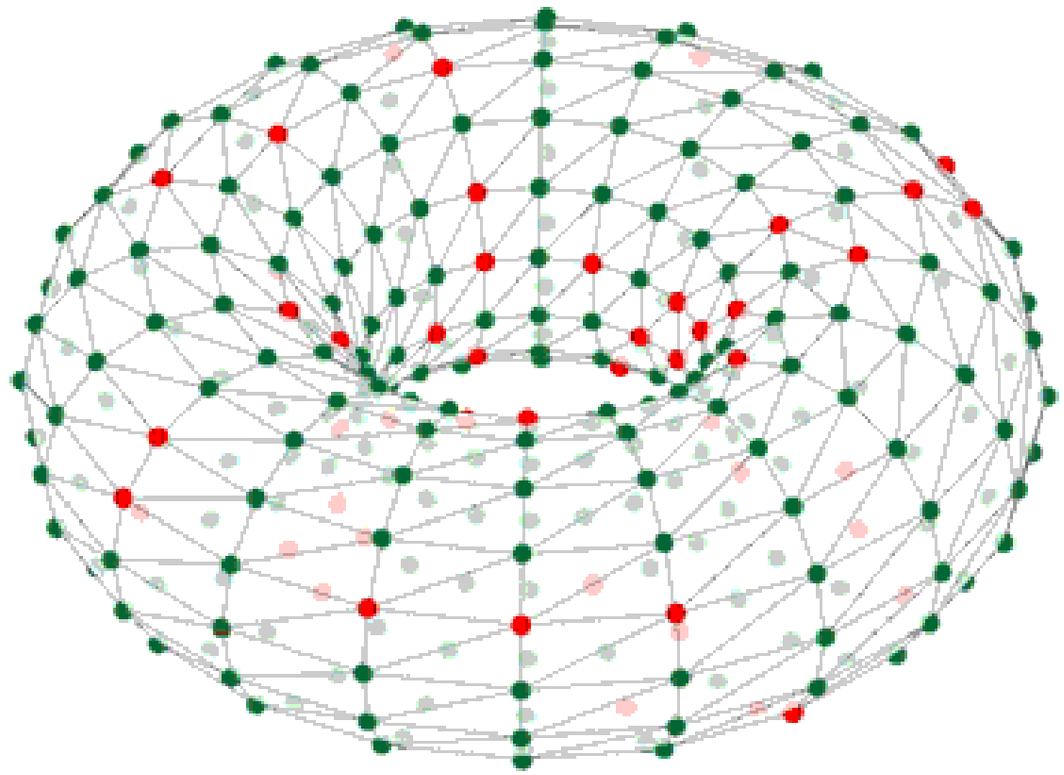
- 1000 neurons per core
- 18000 neurons per chip
- 864000 neurons per board
- 3.1Gbps SATA connections for connecting to other boards
- two 100Mbps Ethernet for control
- max 70W consumption, low temp



Neuromorphic hardware – SpiNNaker

In order to provide fast spike transmission between cores, a proper connectivity method must be exploited.





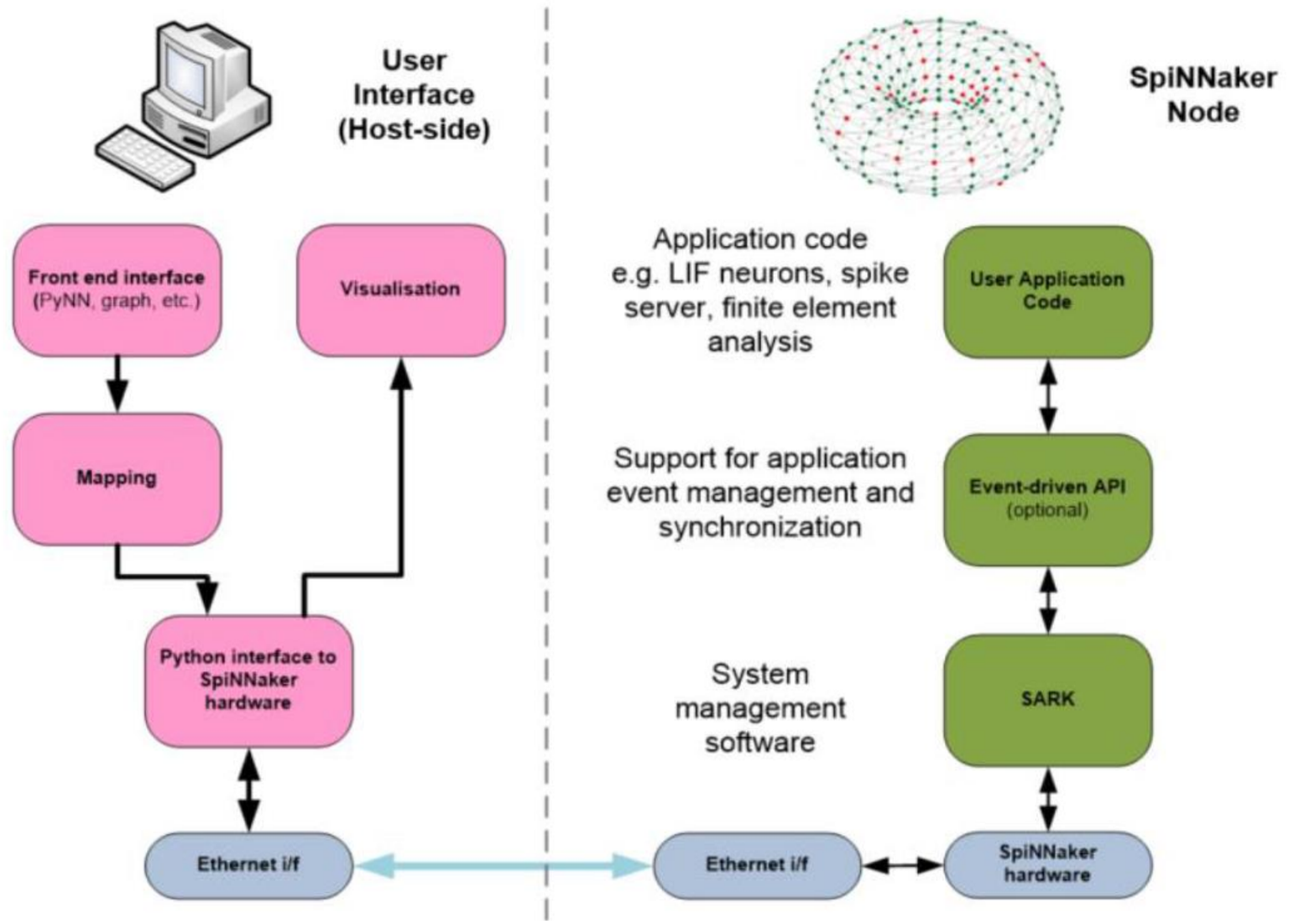
Toroidal connectivity ensures fast spike delivery among chips.

Neuromorphic hardware – SpiNNaker

How do one use these boards? There is a Python library that we can use to set up the network on the SpiNNaker cores: PyNN.

PyNN is a frontend for different neural simulators (including SpiNNaker and NEST)

neuralensemble.org/PyNN

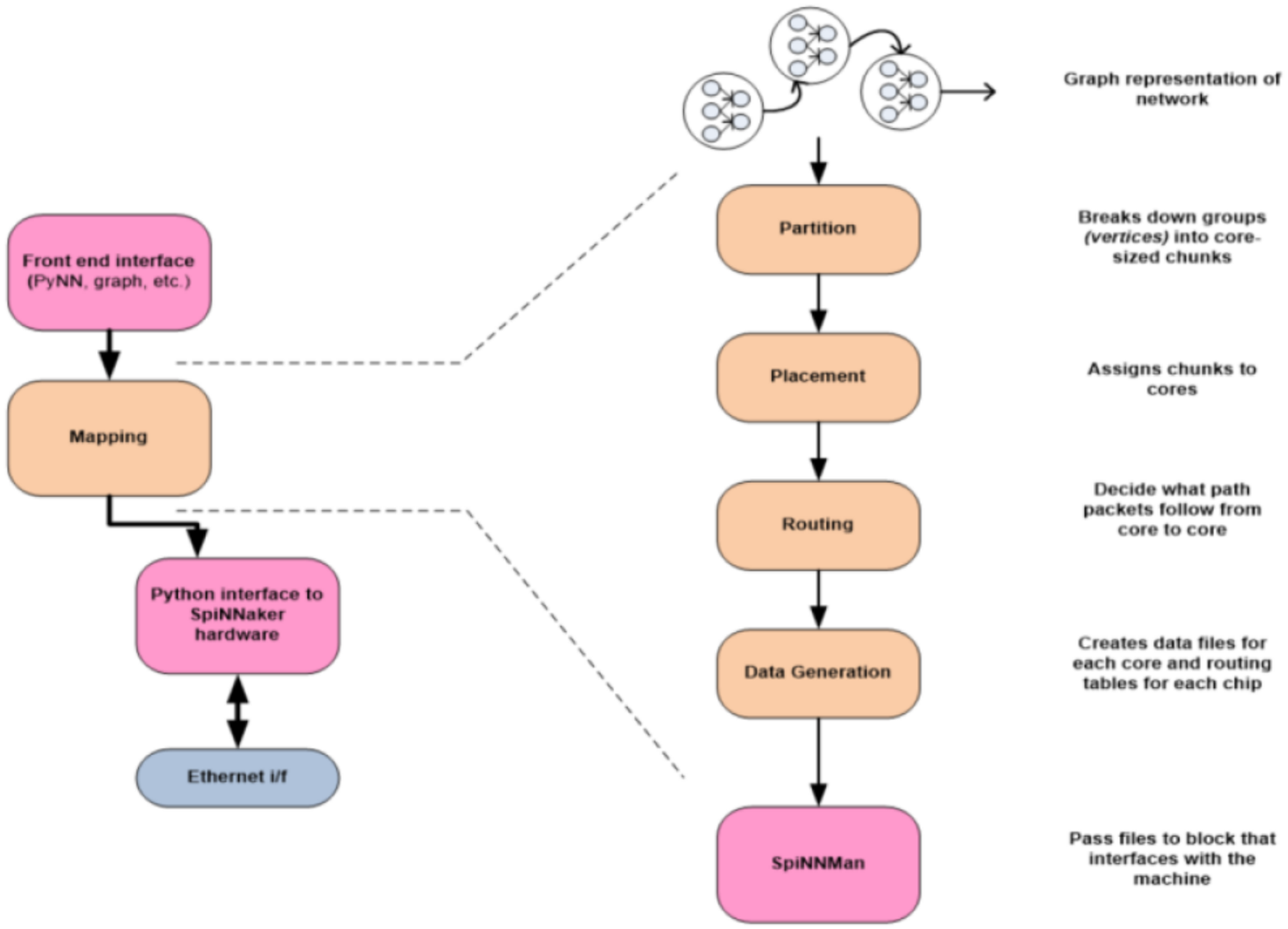


Neuromorphic hardware – SpiNNaker

How do one use these boards? There is a Python library that we can use to set up the network on the SpiNNaker cores: PyNN.

PyNN is a frontend for different neural simulators (including SpiNNaker and NEST)

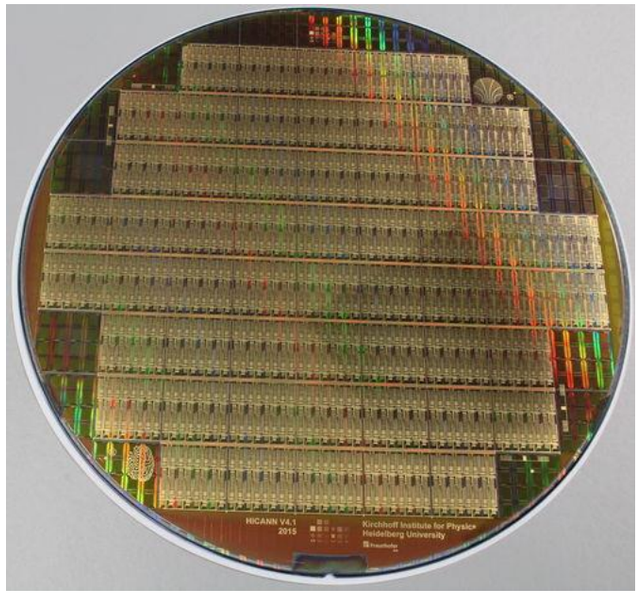
neuralensemble.org/PyNN

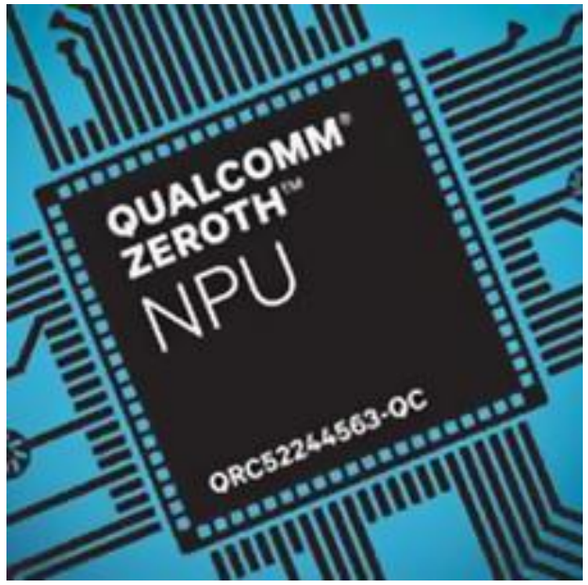


Neuromorphic hardware

SpiNNaker is not the only neuromorphic hardware platform:

Name	Developer	Features
TrueNorth	IBM	Custom processor, 4096 cores with 256 neurons each
BrainScaleS	Heidelberg	Physical model, accelerated simulation time
Brainstorm	Stanford	Physical model, real time
Zeroth	Qualcomm	Deep learning on Snapdragon





Neuromorphic hardware – Final remarks

Pros:

- real time neural simulation
- low power consumption
- portable (can be embedded on robots)

Cons:

- cost, availability
- limited number of neurons and connections by design
- still in development

Outline

1. Introduction
2. Fundamentals of neuroscience
3. Simulating the brain
4. Software and hardware simulations
5. **Robotic applications**

Robotic applications

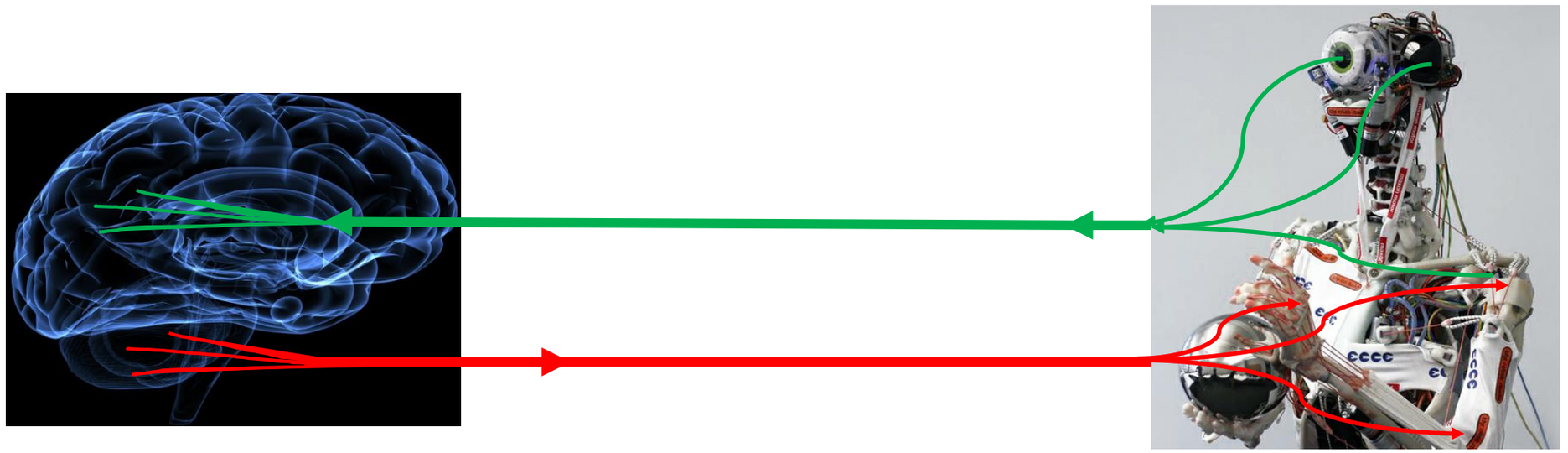
How can we integrate brain models with robotic platforms?

- spiking neural network should be integrated alongside classic robot controllers, relieving them of some computation
- bio-inspired brain models works well for processing of data coming from bio-inspired sensors
- bio-inspired brain models works well for bio-inspired actuators (tendon driven robots, muscle like actuators)
- if connected to a robot, the neural simulation must run in real-time
- if real-time neural simulation is not possible we have to simulate also the robot

The Neurorobotics Platform

Human Brain Project

The Neurorobotics Platform is a simulation toolkit that aims at providing synchronized neural and robotic simulations, and data transfer from robot sensors/actors to brain areas and vice versa.



fortiss

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

FZI

THE BIOROBOTICS INSTITUTE

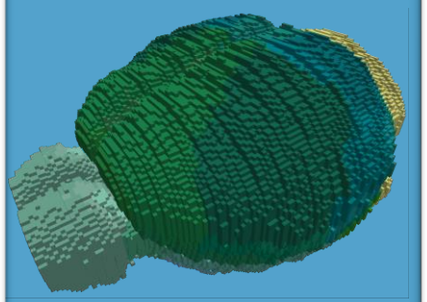
Scuola Superiore Sant'Anna

The Neurorobotics Platform

Human Brain Project

Step 1

Choose desired system

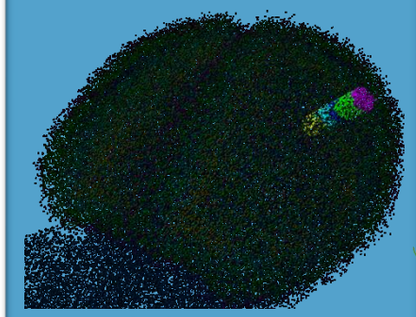


Detailed Brain Model

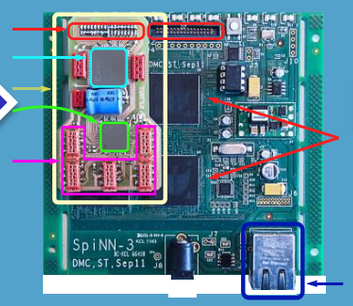
Body/Environment system

Step 2

Choose desired level & platform



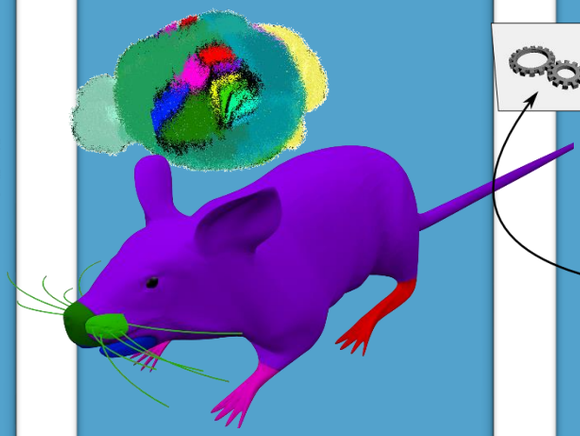
- Detailed neurons
- Point neurons
- Abstract neurons



- HPC simulation
- Analog neuromorphic
- SpiNNaker

Step 3

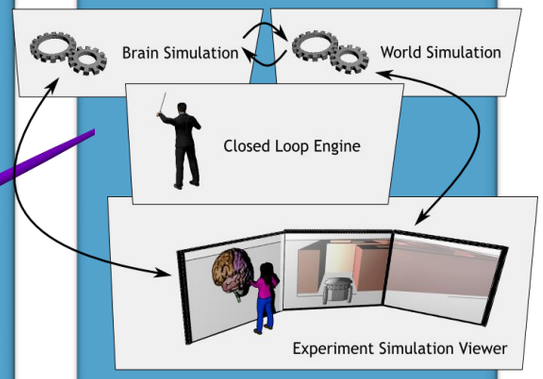
Instrumentation & alignment



- Determine signals
  - Sensors
  - Actuators
  - Motors
- Determine regions
  - Sensory
  - Motor
- Re-instantiate representations for sensors and motors

Step 4

Execution Run the experiment



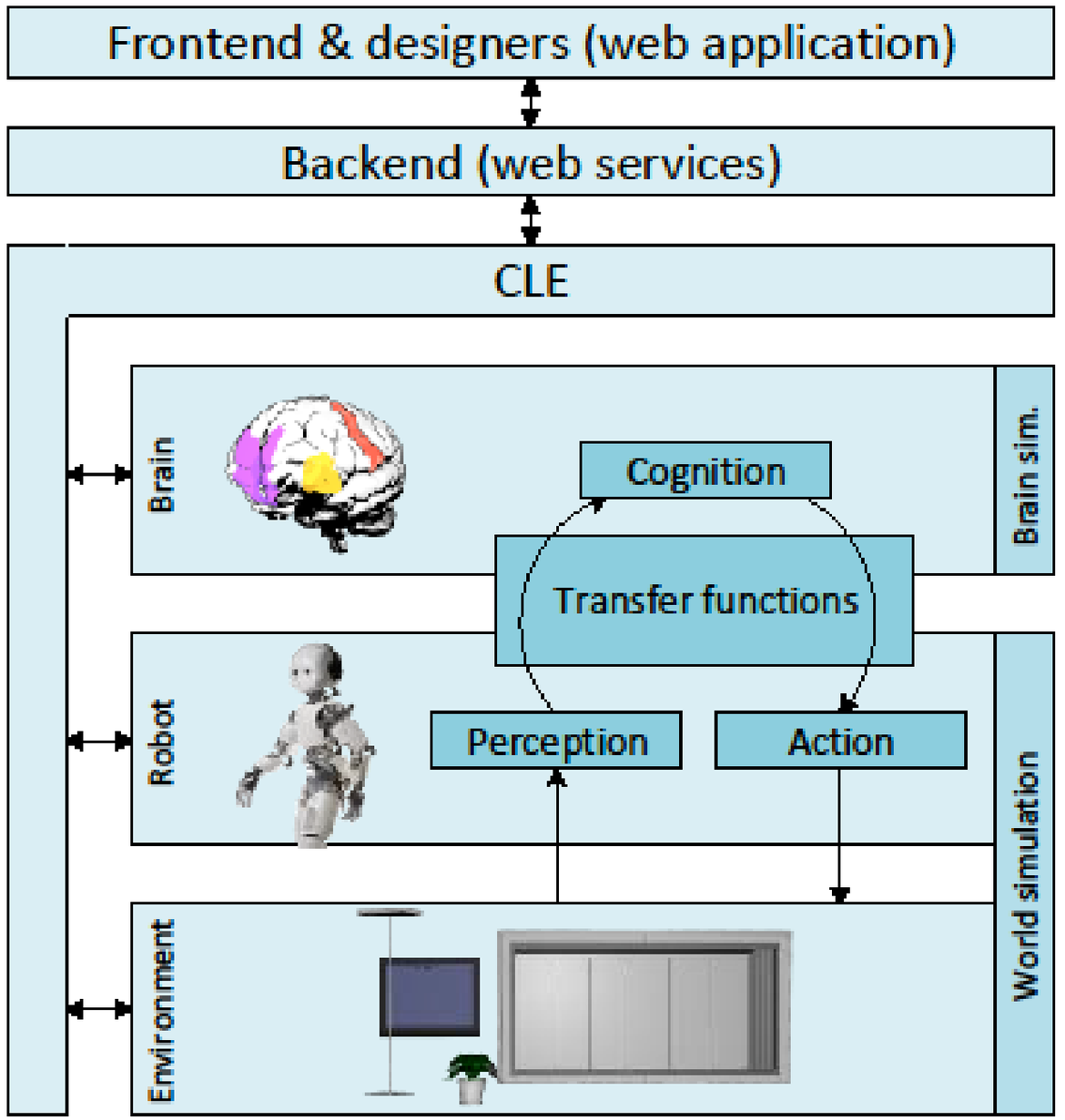
- Interact with users
- Coordinate all models
  - Brain Model
  - Robot Model
  - Environment
- Coordinate simulators

The Neurorobotics Platform

Human Brain Project

Key features:

- action-perception loop provided by transfer functions
- neural and physical simulations synchronization
- complete framework where experiments can be constructed from scratch
- web interface

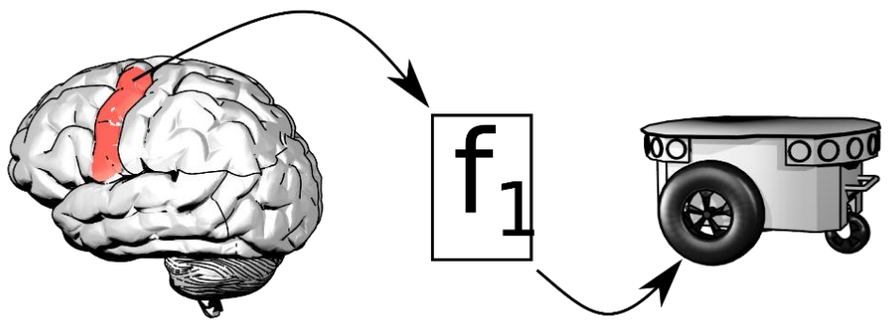


The Neurorobotics Platform

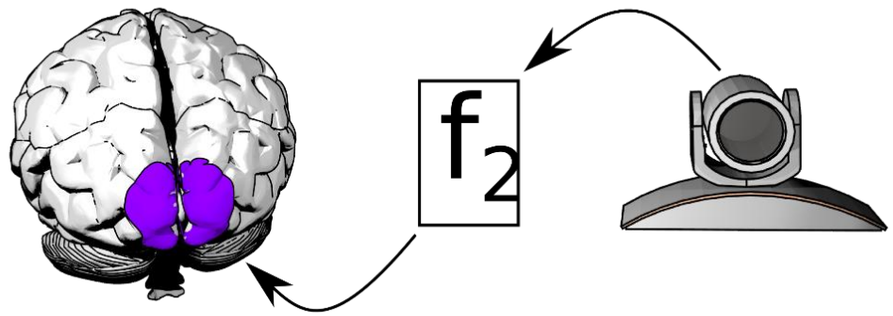
Human Brain Project

The main idea behind the Neurorobotics Platform is the *action-perception closed loop*, operated by **transfer functions**.

- *robot to neuron* transfer functions translate sensory information into spikes and current amplitudes



- *neuron to robot* transfer functions take measurements on the neural network (spike rate, membrane potential) and transform them into robot commands



The Neurorobotics Platform

Human Brain Project

The transfer functions connect the neural and physical simulators.

The **neural simulation** is provided by NEST, through the PyNN interface.

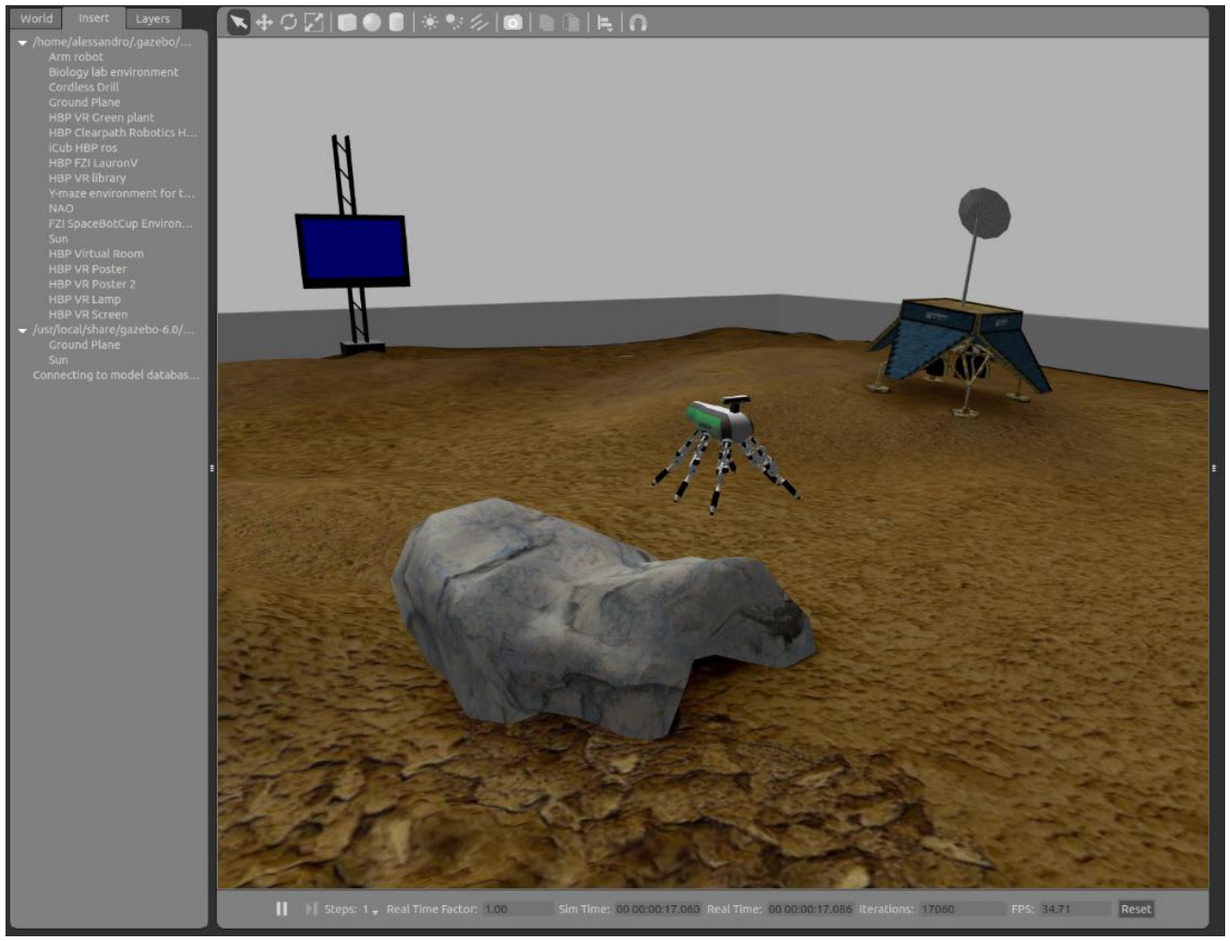
The **physical and robotic simulations** are provided by Gazebo, via the ROS middleware.

GAZEBO

gazebosim.org

ROS

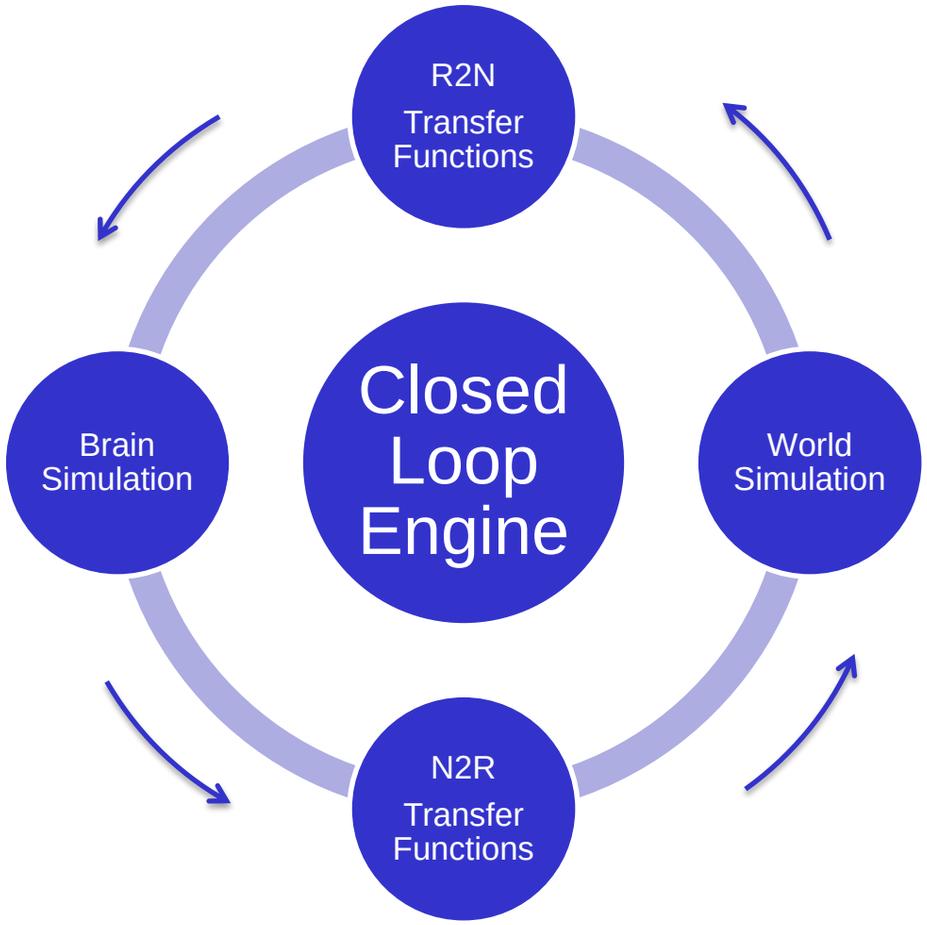
ros.org

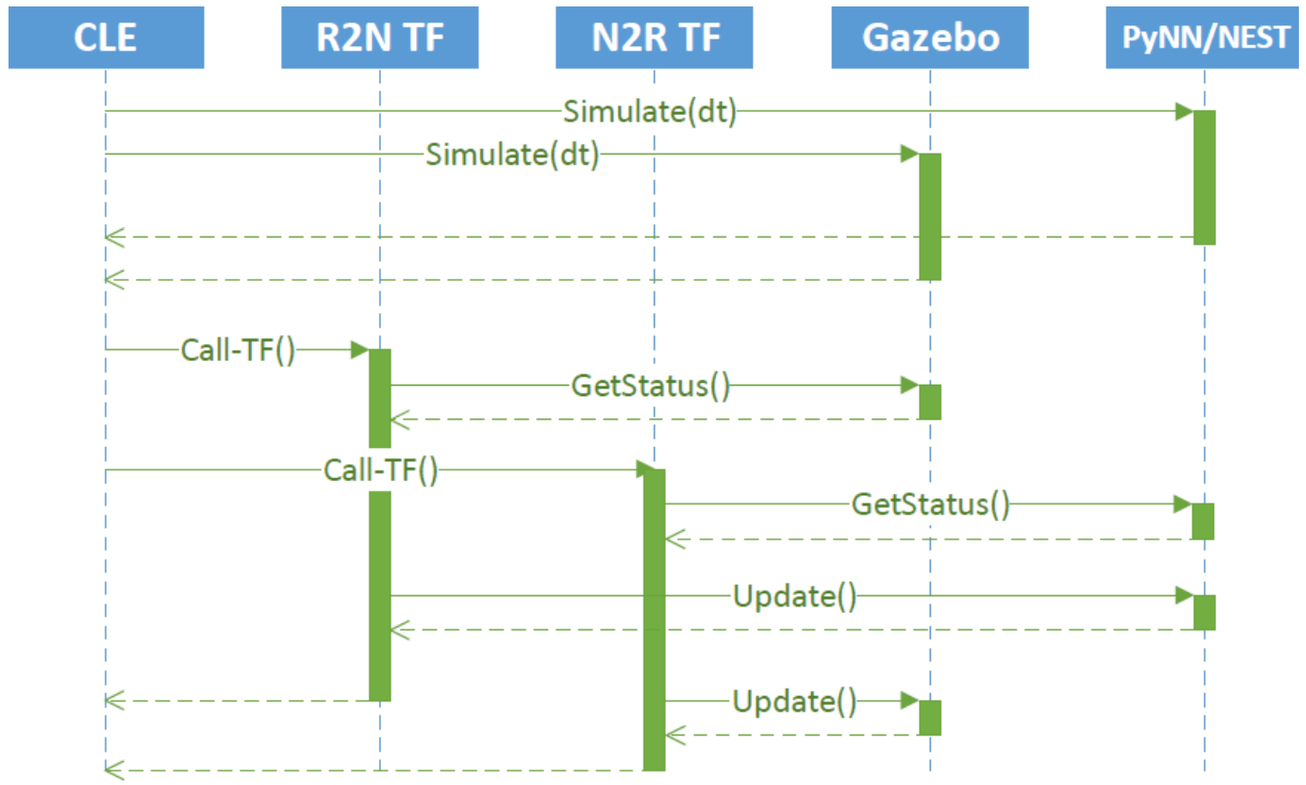


The Neurorobotics Platform

Human Brain Project

The **Closed Loop Engine** is responsible for the synchronization and data exchange between the two simulations.



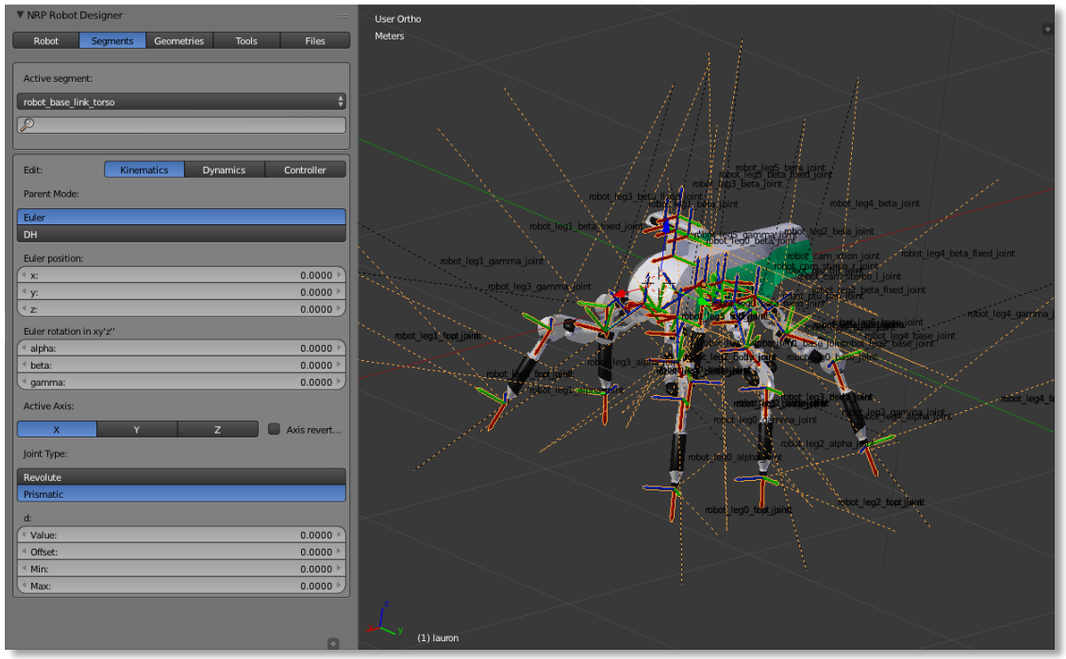


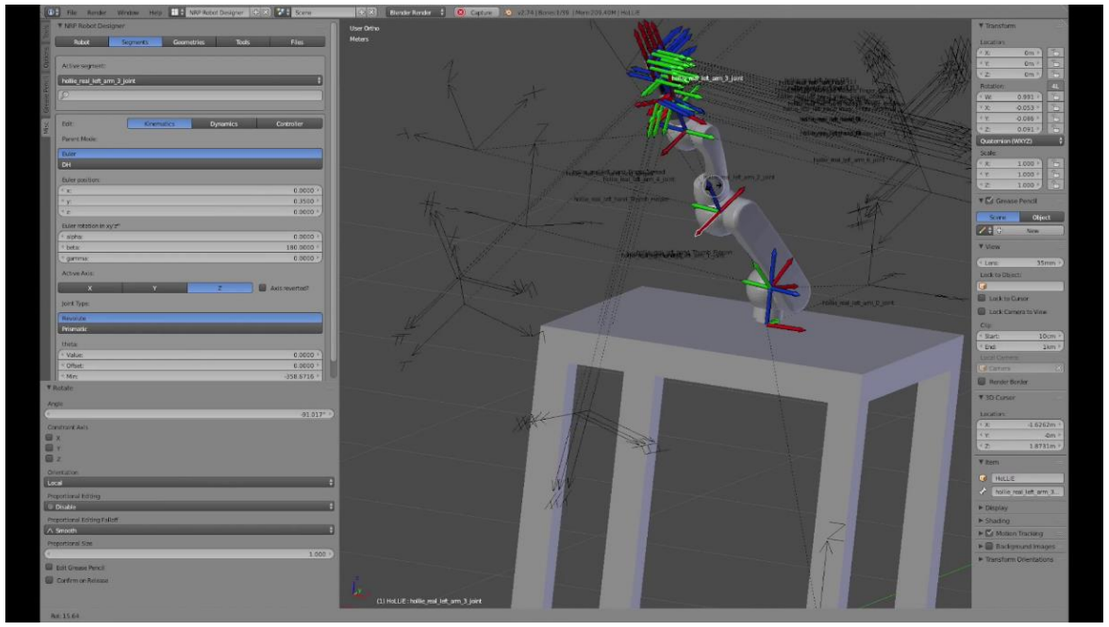
The Neurorobotics Platform: Demo

The Neurorobotics Platform: Designers

In order to let the user design his own experiments, three different designers are being developed inside the NRP:

Through the use of the **Robot Designer**, the user is able to create his own robot either by designing it from scratch or by editing an existing one.



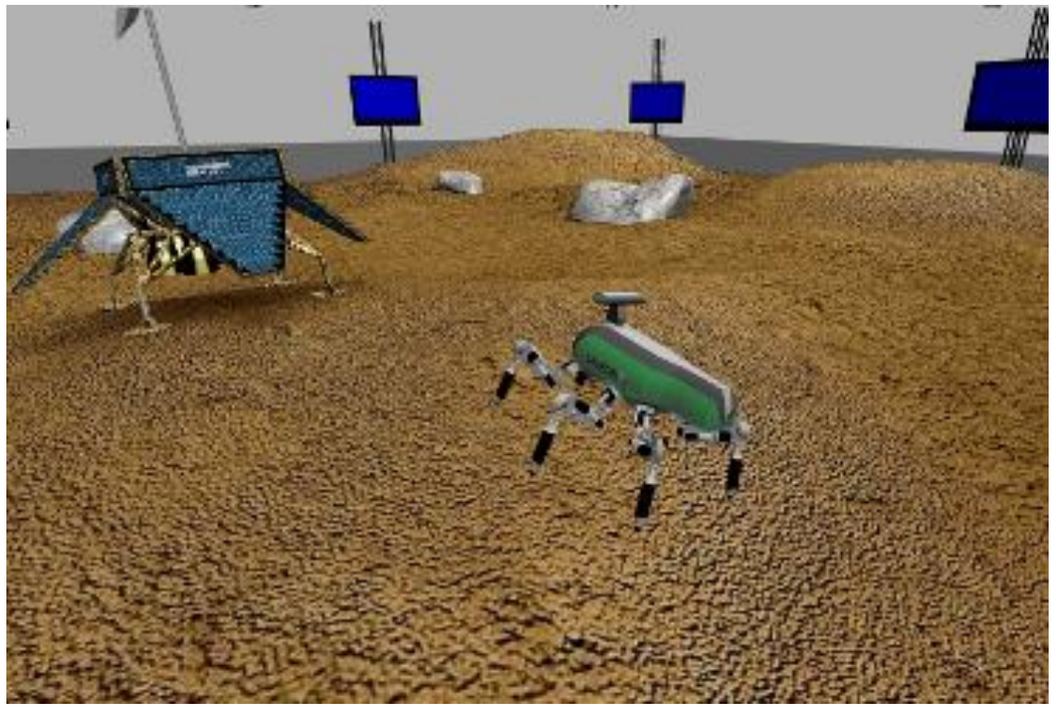


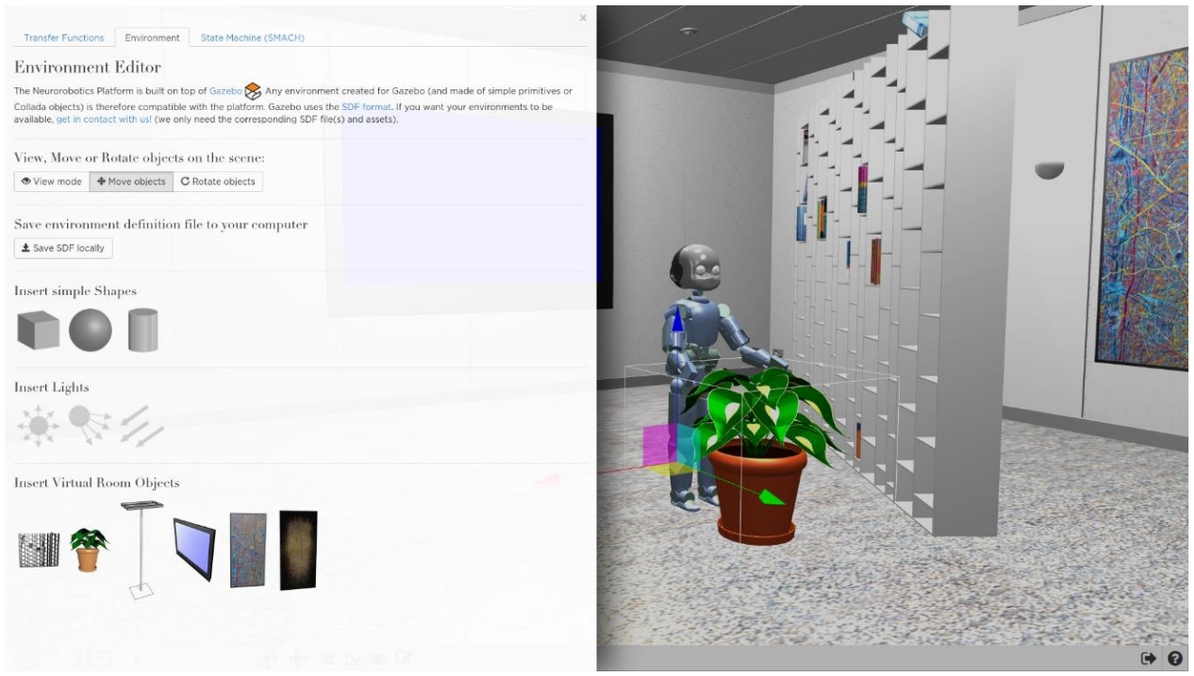
Aside from the appearance of the robot, kinematic parameters can be edited and actuators and sensor can be added.

The Neurorobotics Platform: Designers

In order to let the user design his own experiments, three different designers are being developed inside the NRP:

The aim of the **Environment Designer** is to create an environment in which the experiment will run.



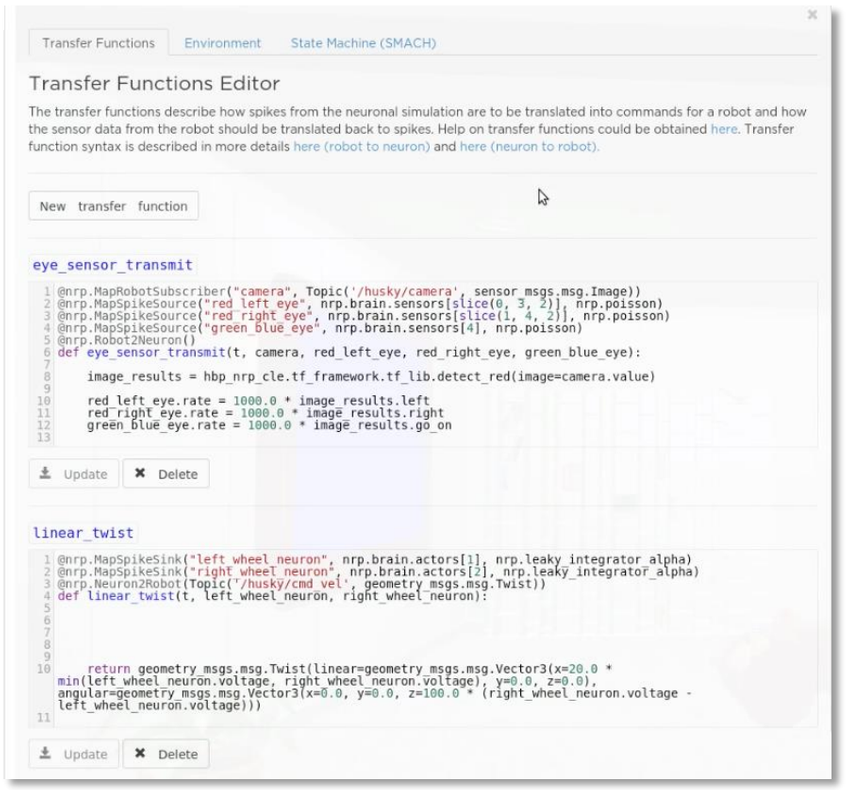


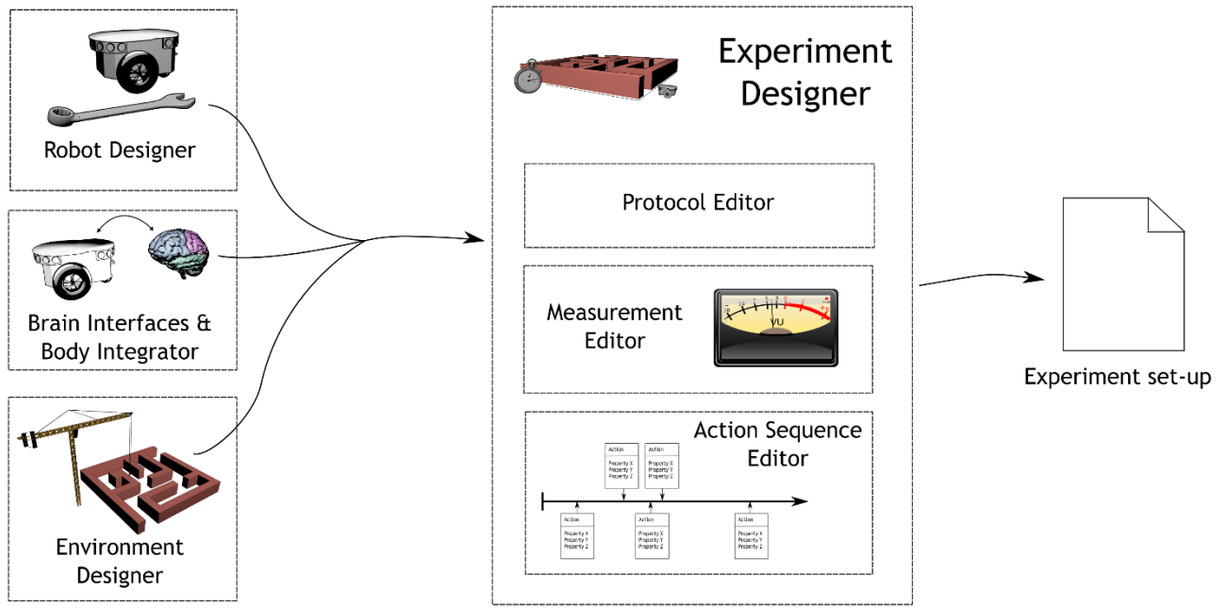
The environment can be modelled from scratch or built from objects present in a predefined library.

The Neurorobotics Platform: Designers

In order to let the user design his own experiments, three different designers are being developed inside the NRP:

Using the **Experiment Designer** the user is able to design an experiment.

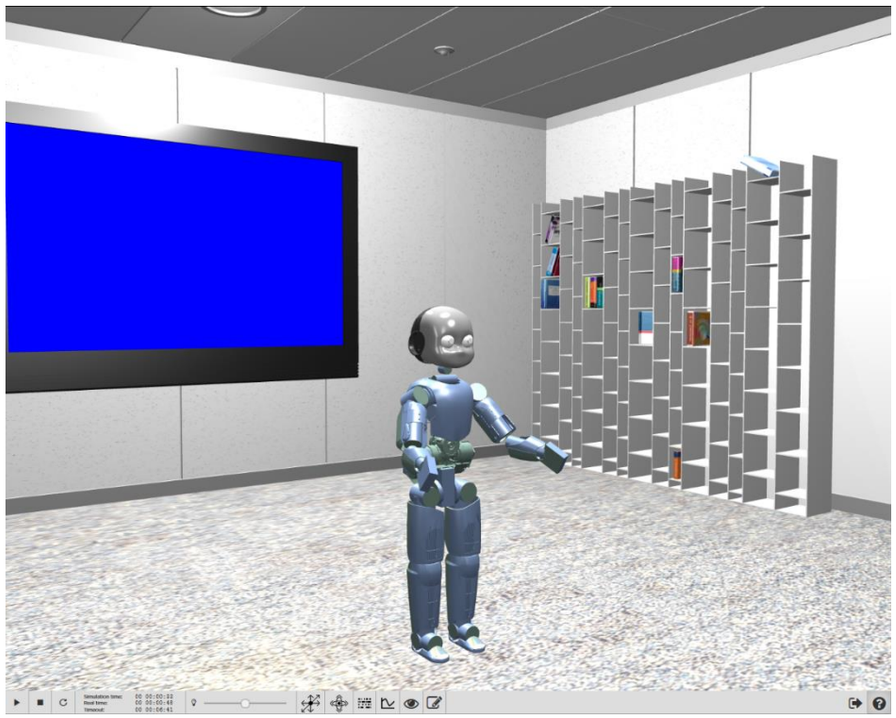


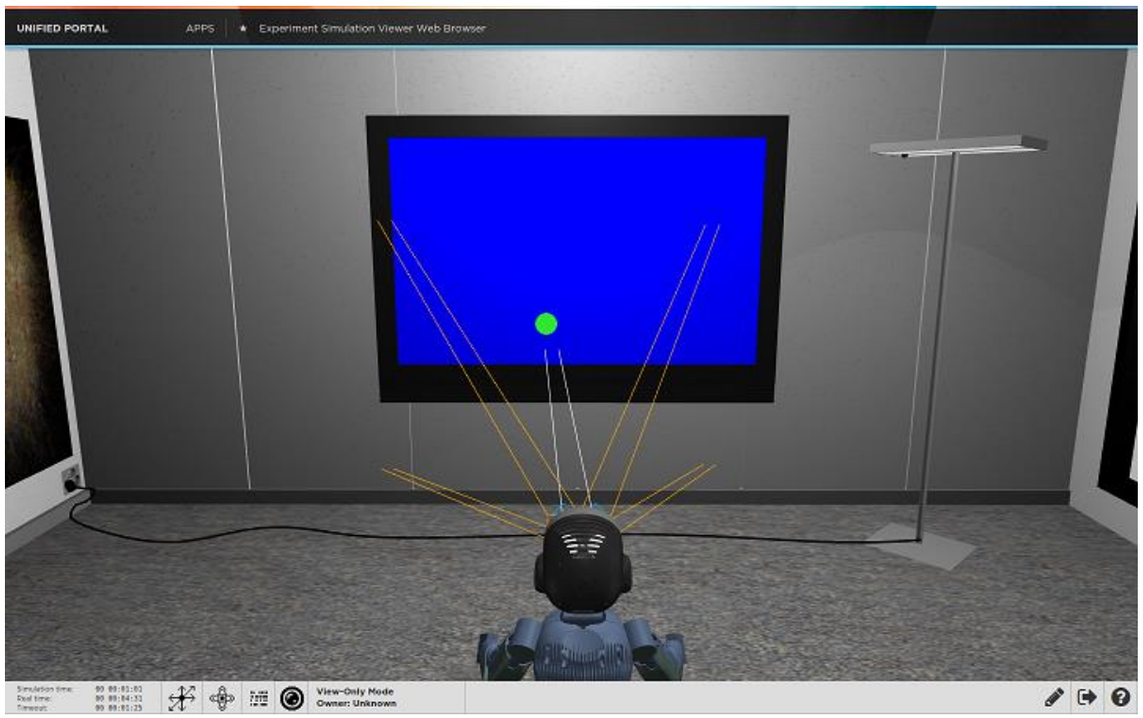


An experiment is composed of: robot, environment, brain model, transfer functions and a collection of events (i.e. time triggered actions that affect the environment).

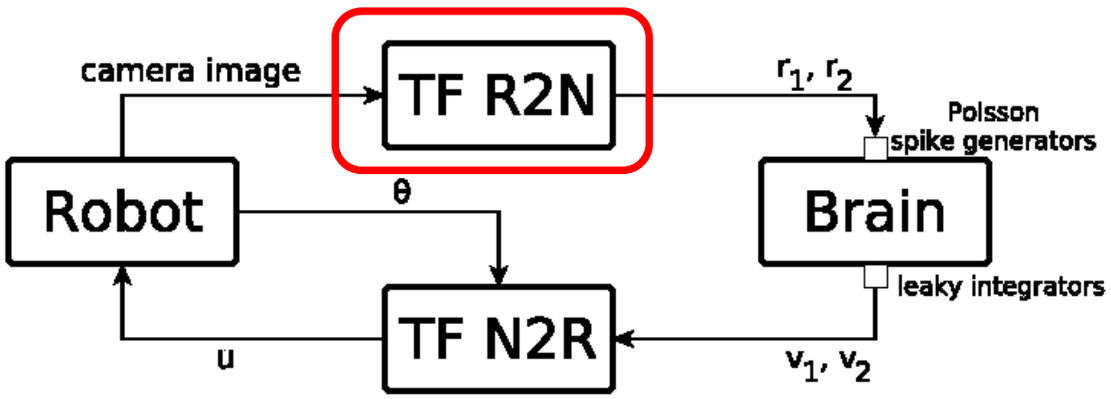
iCub visual tracking in the NRP

A humanoid robot (iCub) stands in front of a virtual screen and follows a moving target using only eye movements generated through a spiking neural network controller.



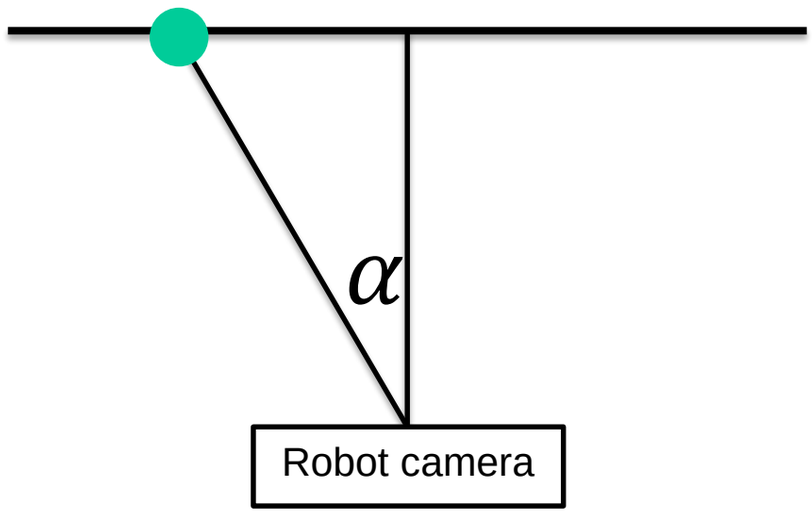


iCub visual tracking in the NRP



To implement such a model in the NRP, the target extraction must be implemented as Robot to Neuron transfer function, while the controller must be implemented as a brain model plus an appropriate Neuron to Robot transfer function.

Robot to neuron transfer function:

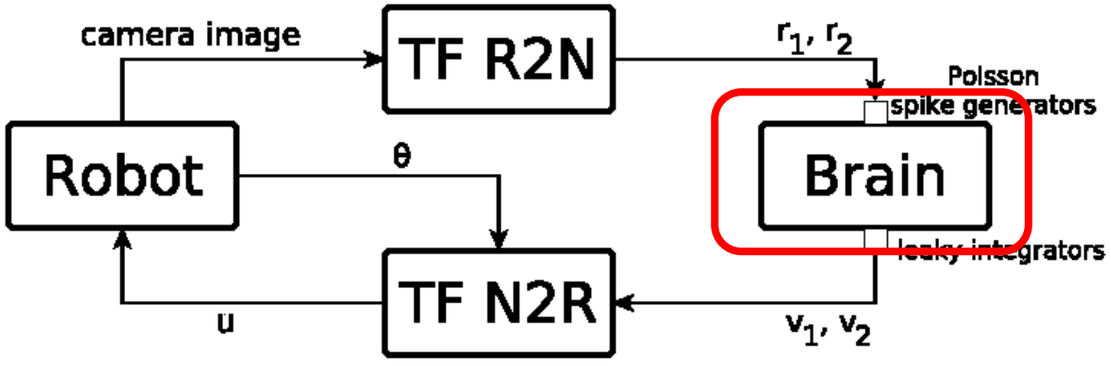


$$r = \frac{1}{1 - e^{\alpha}}$$

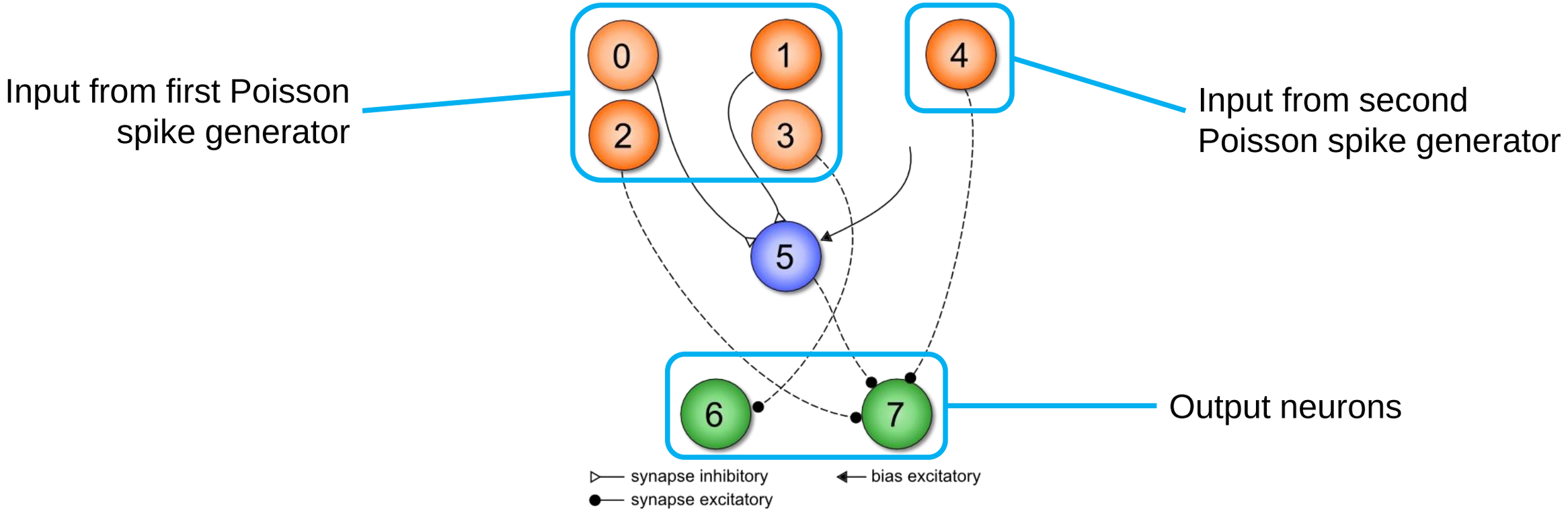
$$r_1 = 1000 \cdot r$$
$$r_2 = 1000 \cdot (1 - r)$$

Poisson spike generator rates

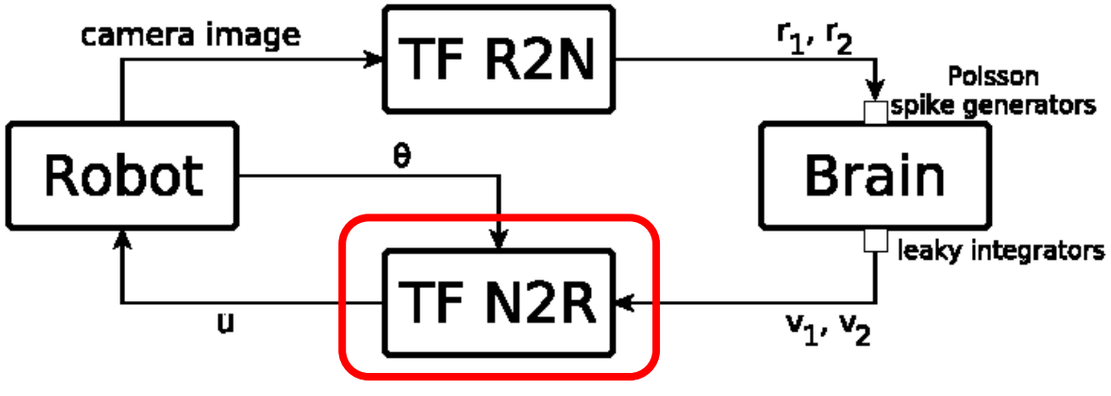
iCub visual tracking in the NRP



The **brain model** was implemented as a network of 8 adaptive exponential integrate and fire neurons.



iCub visual tracking in the NRP



The **neuron to robot** transfer function generates position motor commands for the eye version, using output from the brain and the encoder value.

$$u = \theta + f(v_1, v_2)$$

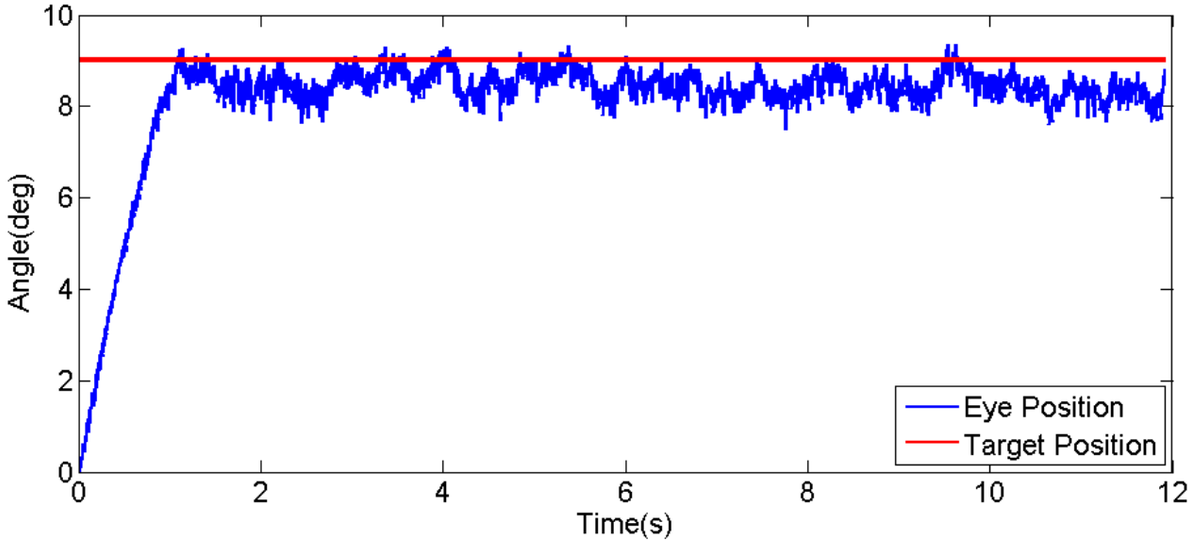
where v_i is the membrane potential of leaky integrator *i* and

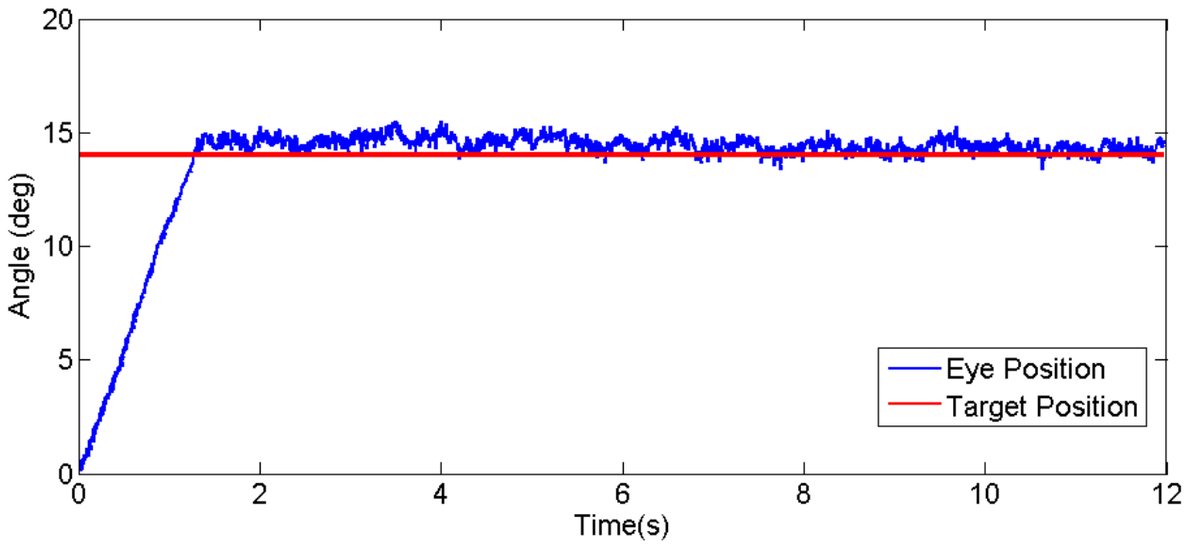
$$f(v_1, v_2) = k - 2 \cdot \frac{v_2 - v_1 + 0.03}{0.09} \cdot k$$

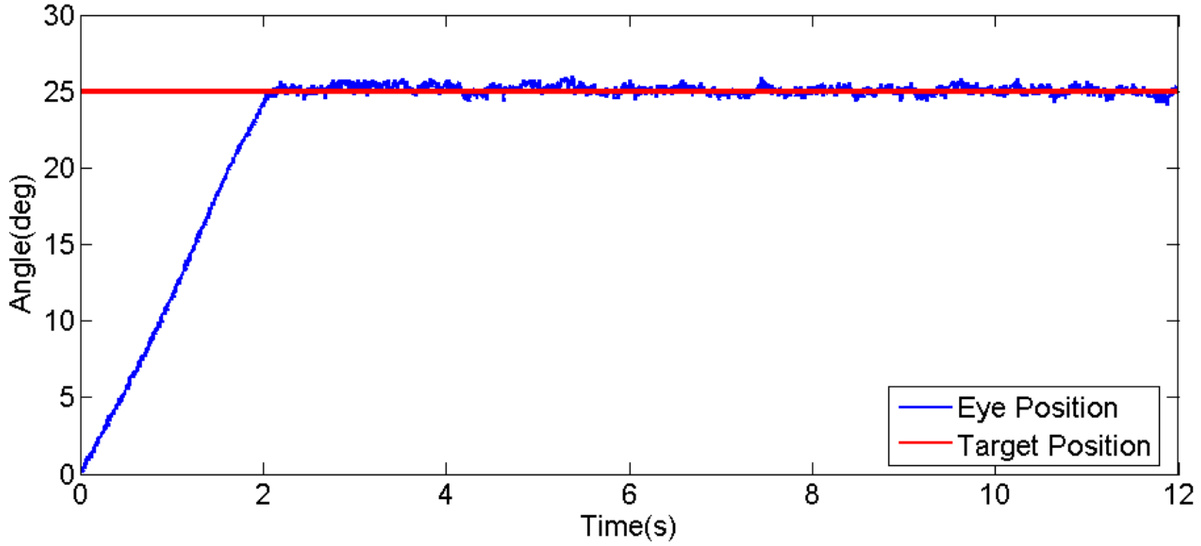
where *k* is the absolute maximum eye displacement at each simulation step

iCub visual tracking in the NRP: Results

Step response towards a rotation distance of 9, 14, and 15 degrees.
The response time is dependent on the motion amplitude (from less than 1 to almost 2 seconds).

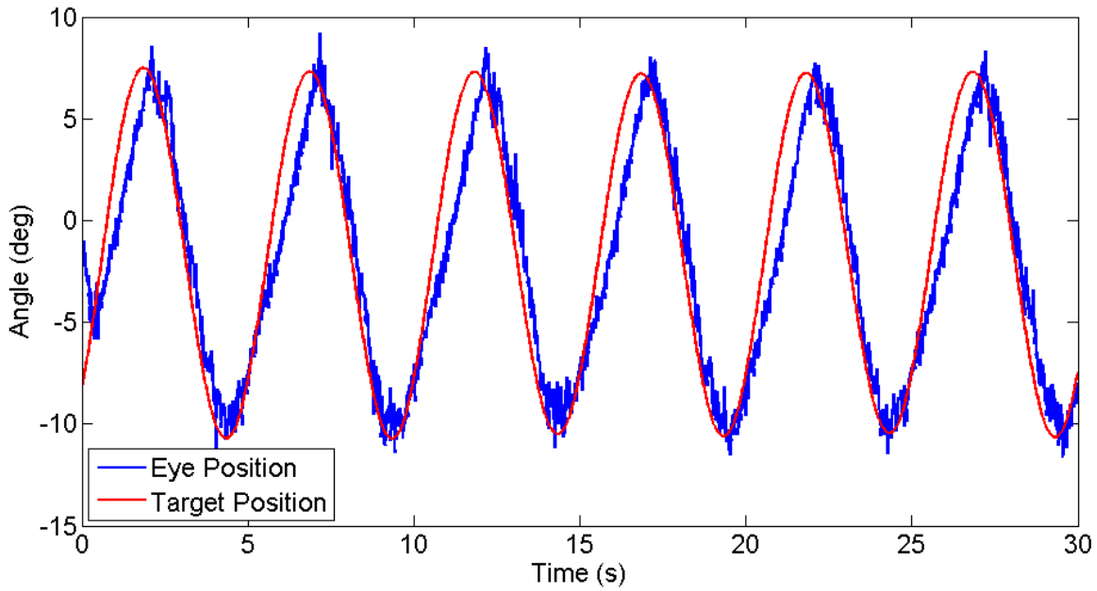


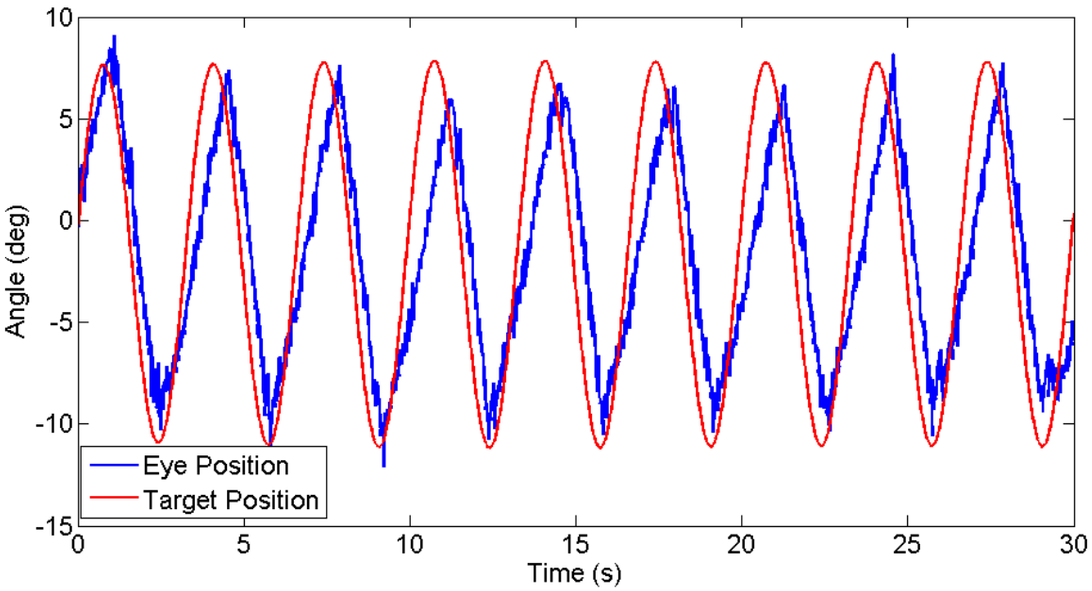




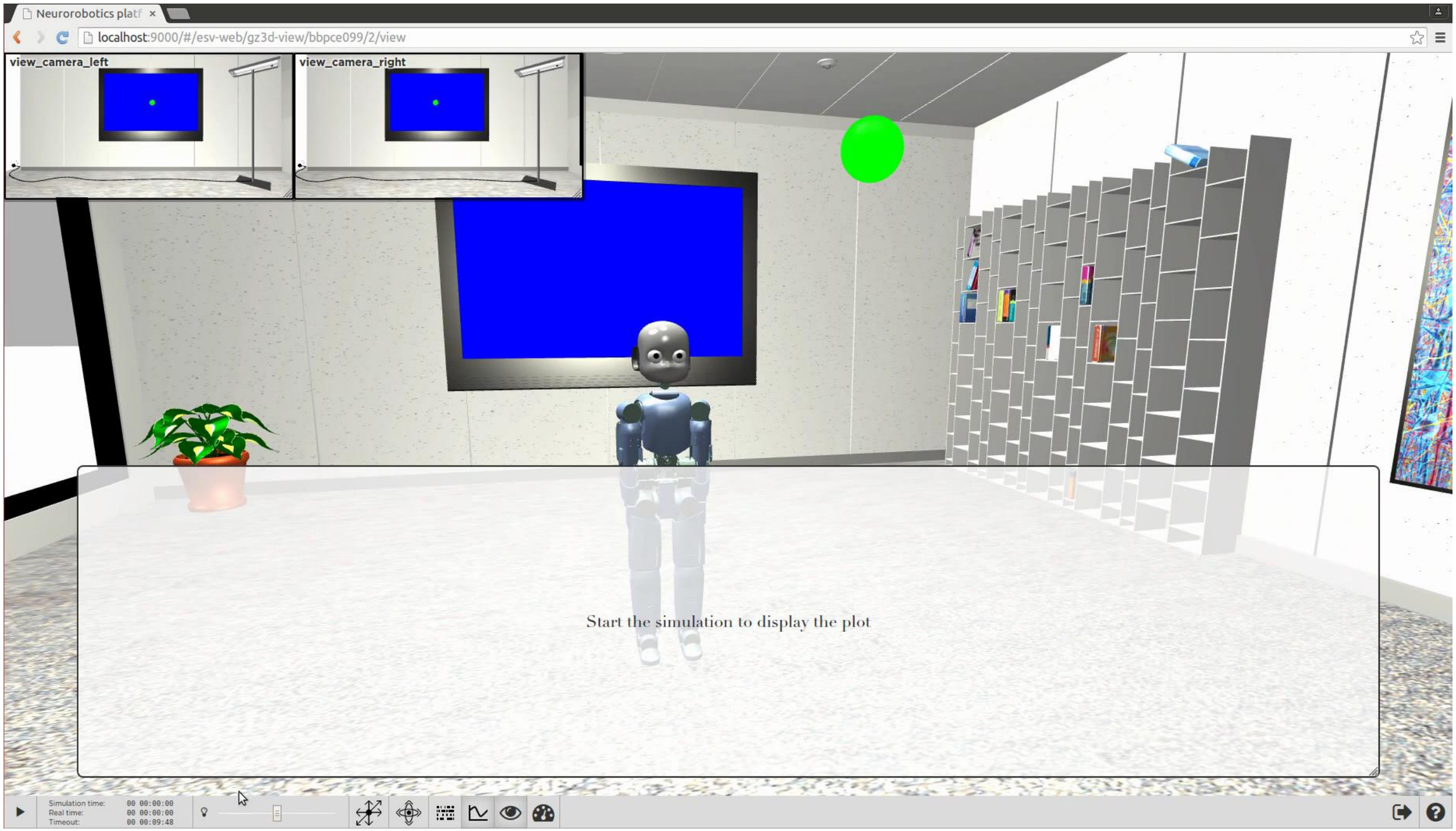
iCub visual tracking in the NRP: Results

Visual tracking of a moving object moving with a sinusoidal motion at various frequencies (from 0.1Hz to 0.3Hz) and a peak-to-peak amplitude of 18 degrees.





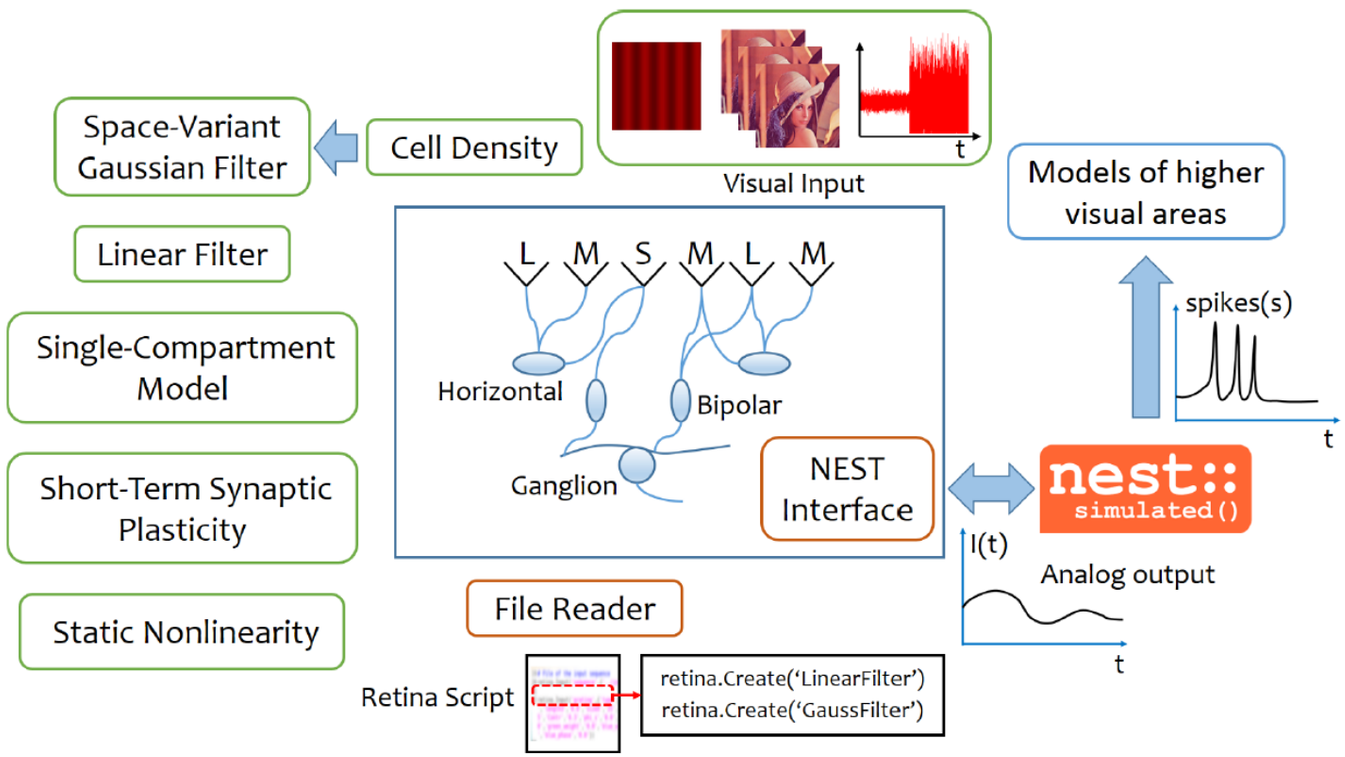
iCub visual tracking in the NRP: Demo



Retinal visual tracking

In this work we integrated bio-inspired sensing with spiking neural network in order to perform a visual tracking task.

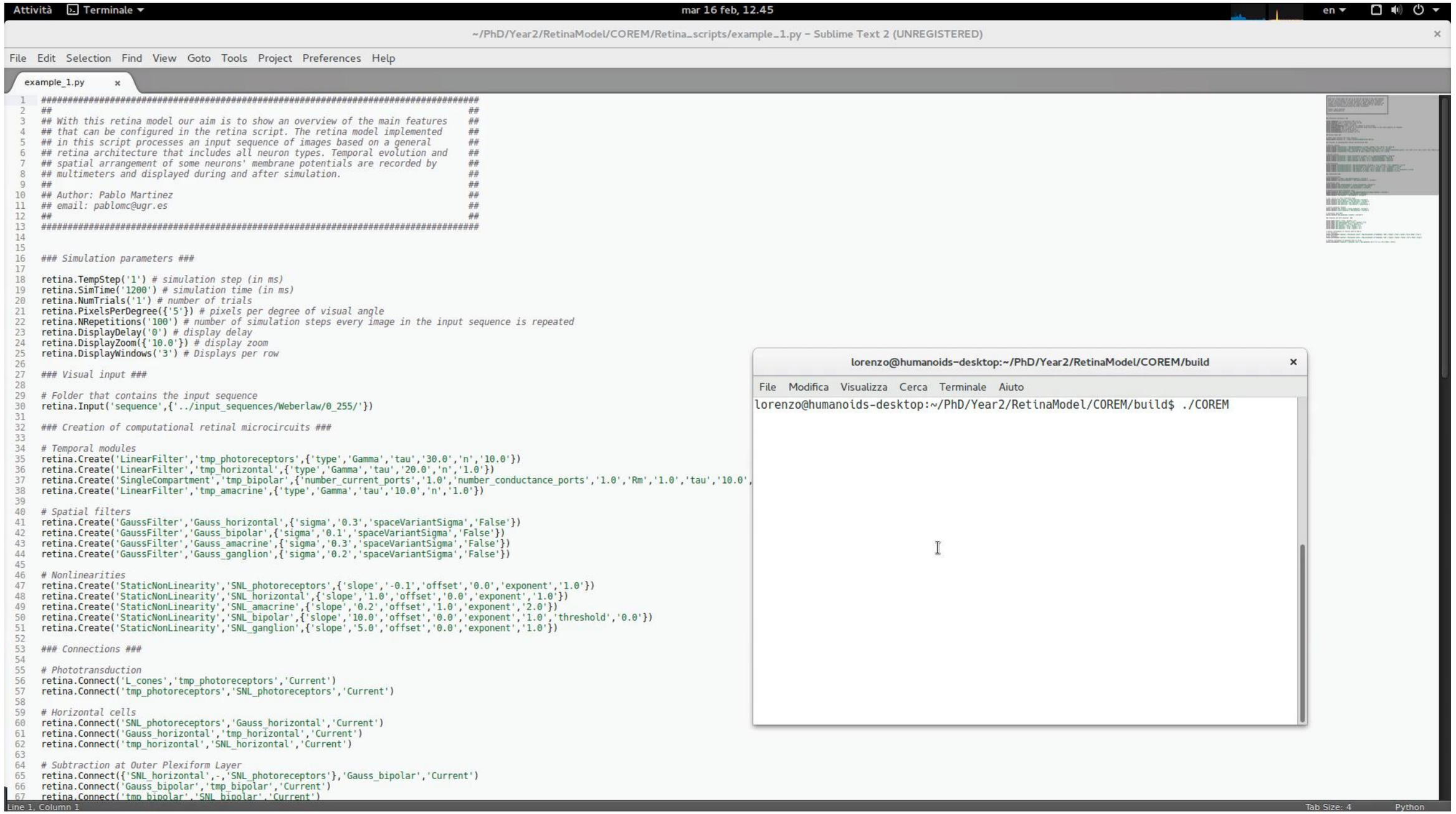
We used the same setup as before, inside the NRP, where we also integrated a retina simulation as a robot to neuron transfer function.



COREM simulator, developed by University of Granada.

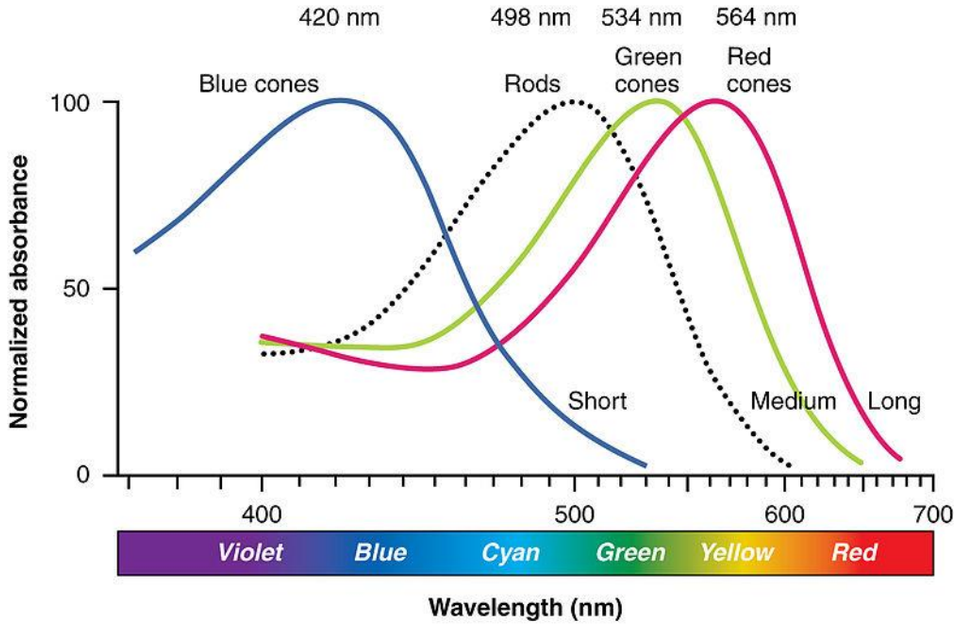
- custom retina models
- based on linear/non-linear analysis
- produces an output compatible with NEST, but not spiking

Retinal visual tracking



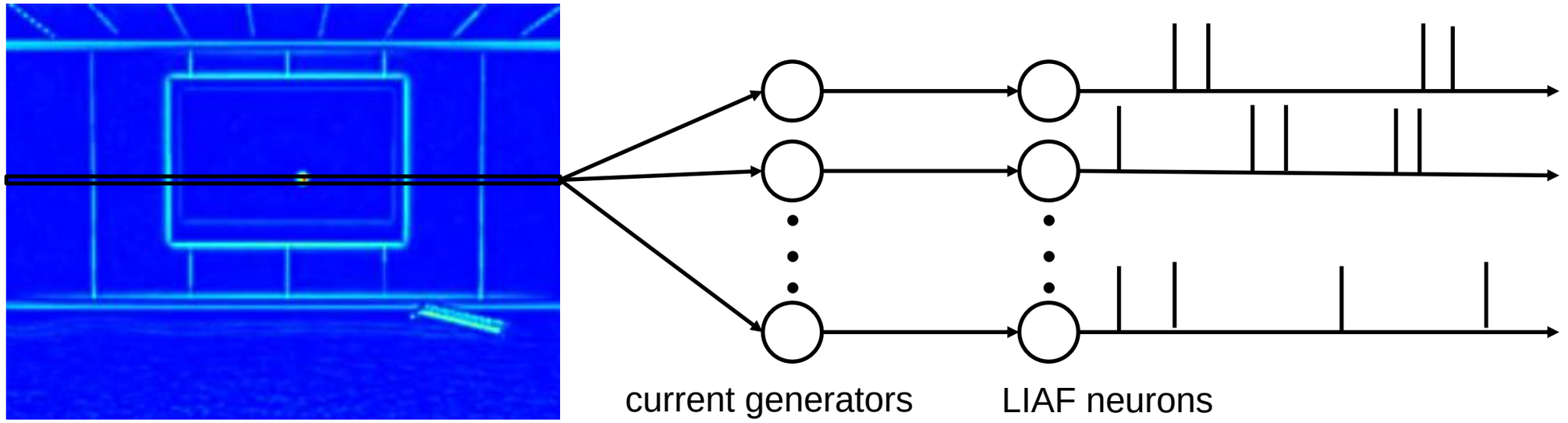
Retinal visual tracking

At first we performed a target detection via retinal image processing.

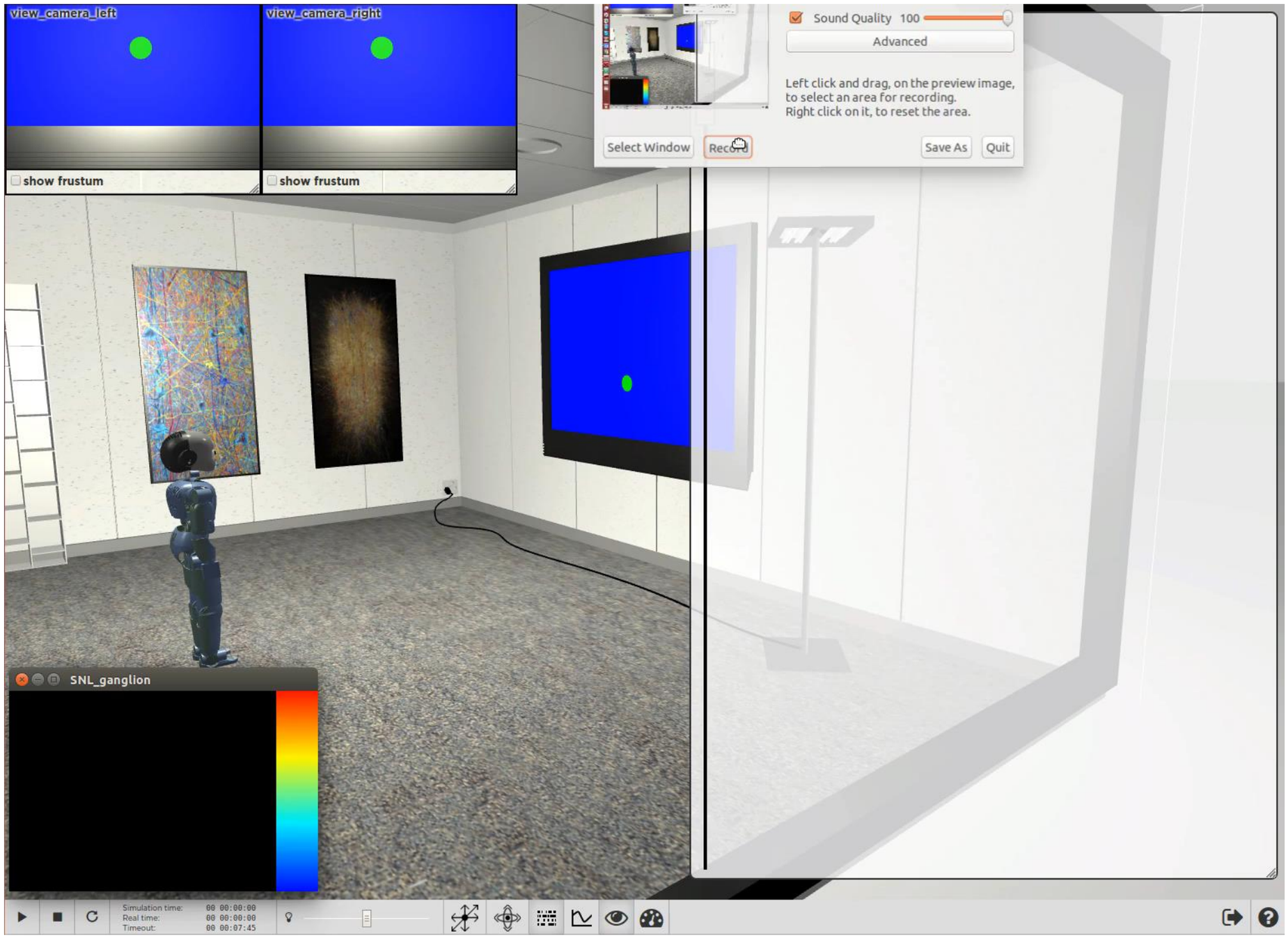


We used a complete retina model, but only with the pathway coming from M-cones (more sensitive to green).

Analogue output from the ganglion cells is sent to a LIAF neuron layer via current generators devices.

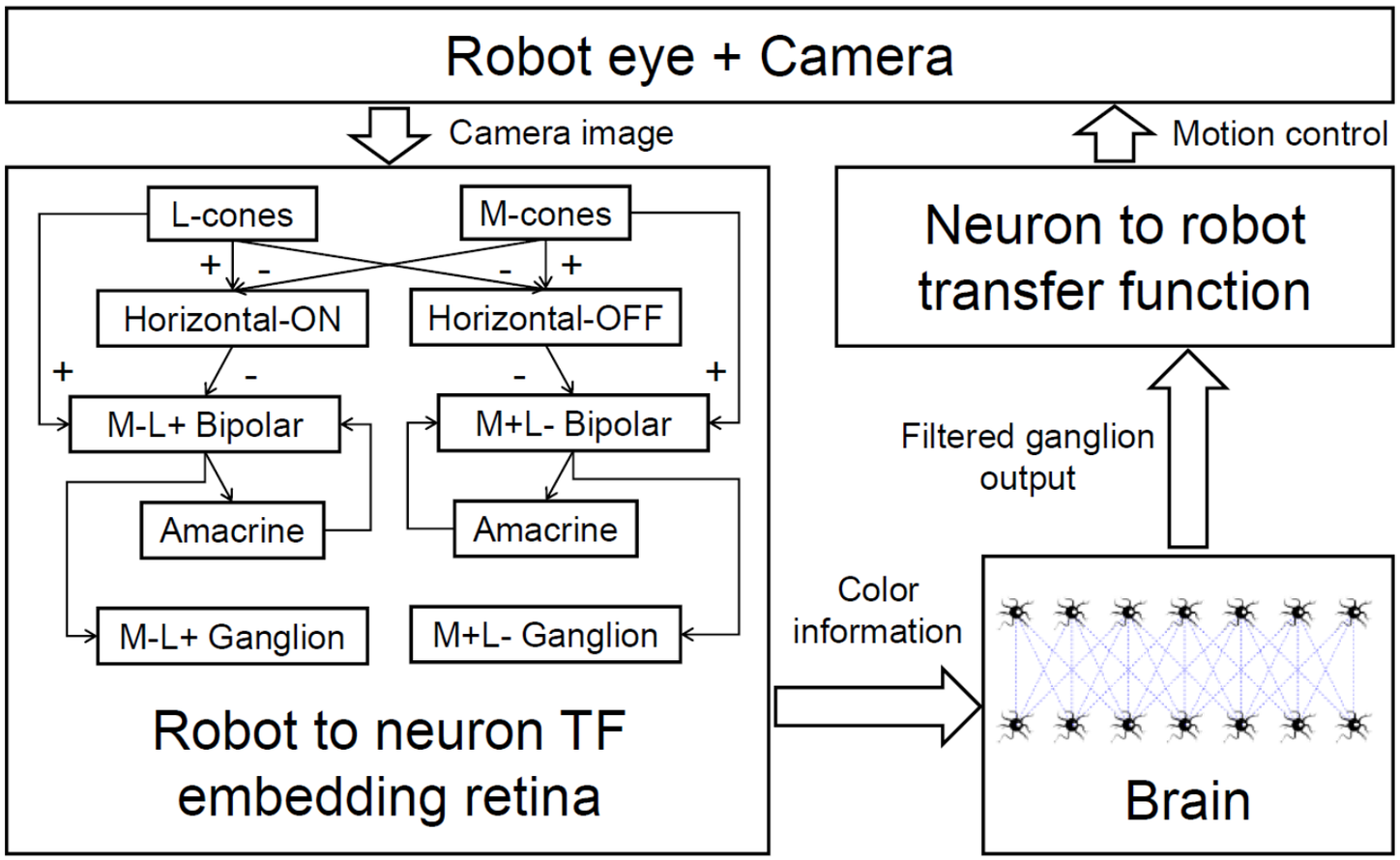


Retinal visual tracking

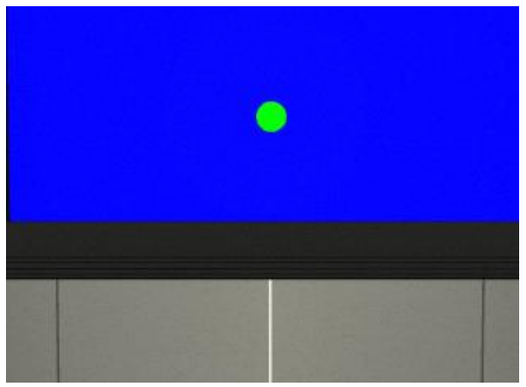


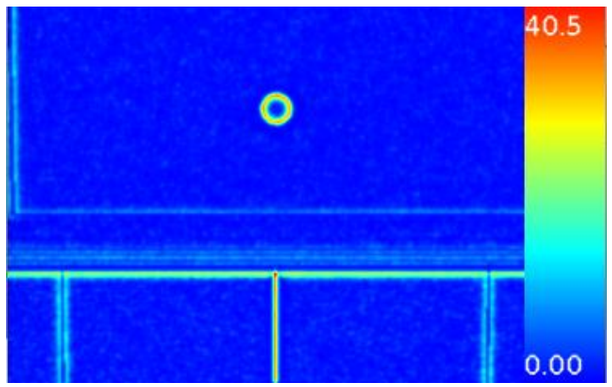
Retinal visual tracking

Then, we switched to a more sophisticated retina model, based on red-green opponency.



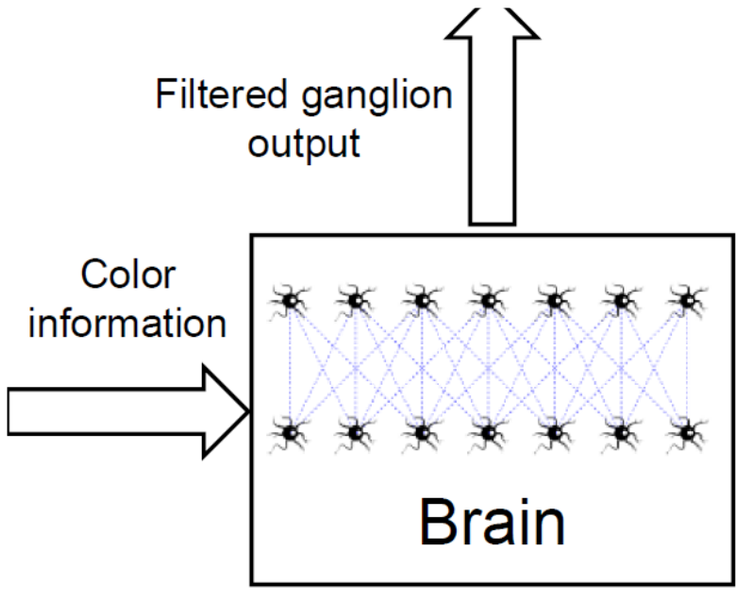
The output value of this retina model is higher for edges of the target.

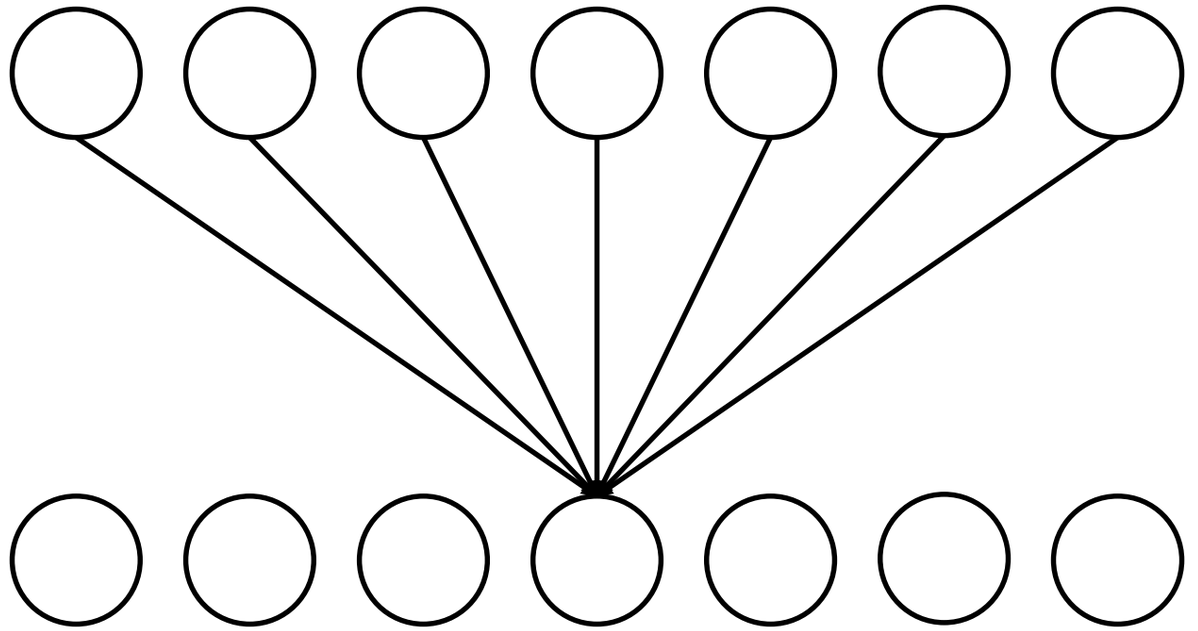




Retinal visual tracking

In order to filter out the noise from the ganglion output and retain only the target information, a two layer spiking neural network was used.

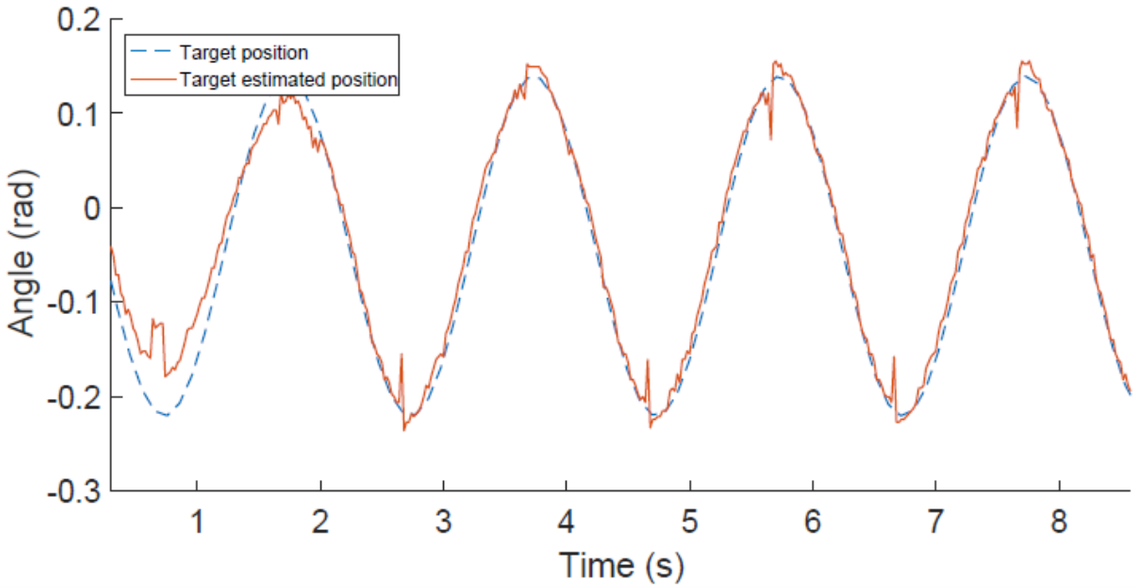


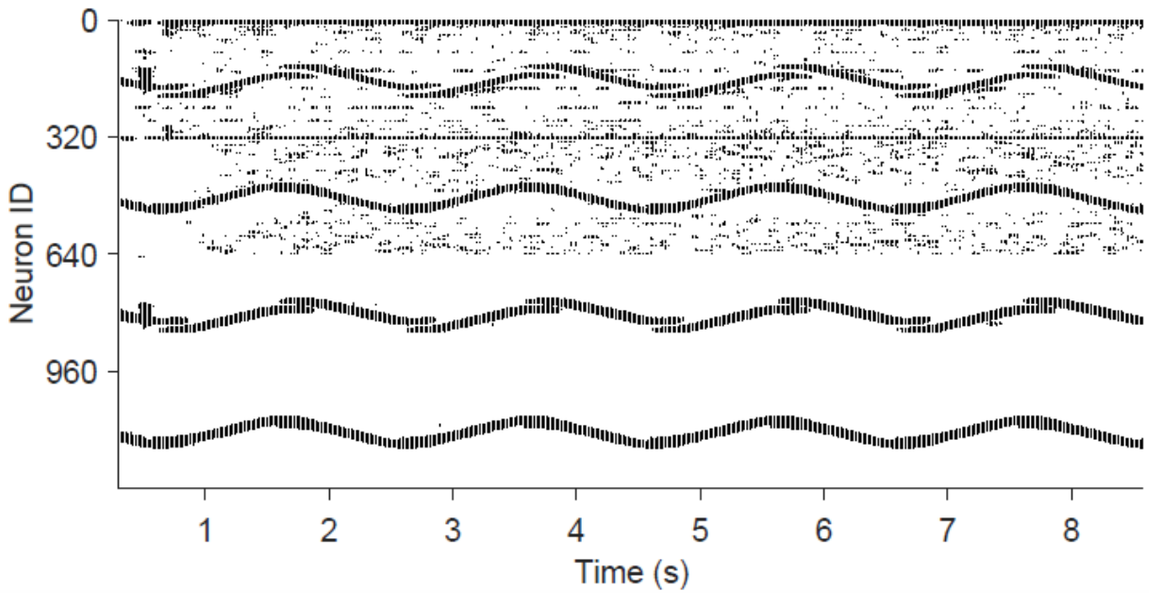


The first layer is a current to spike converter. Neurons in the second layer receives spikes from a *receptive field* of 7 neurons (pixels).

Retinal visual tracking

Using output of the neural network we can estimate the target centroid and use this information to generate motor commands for the robot eye.

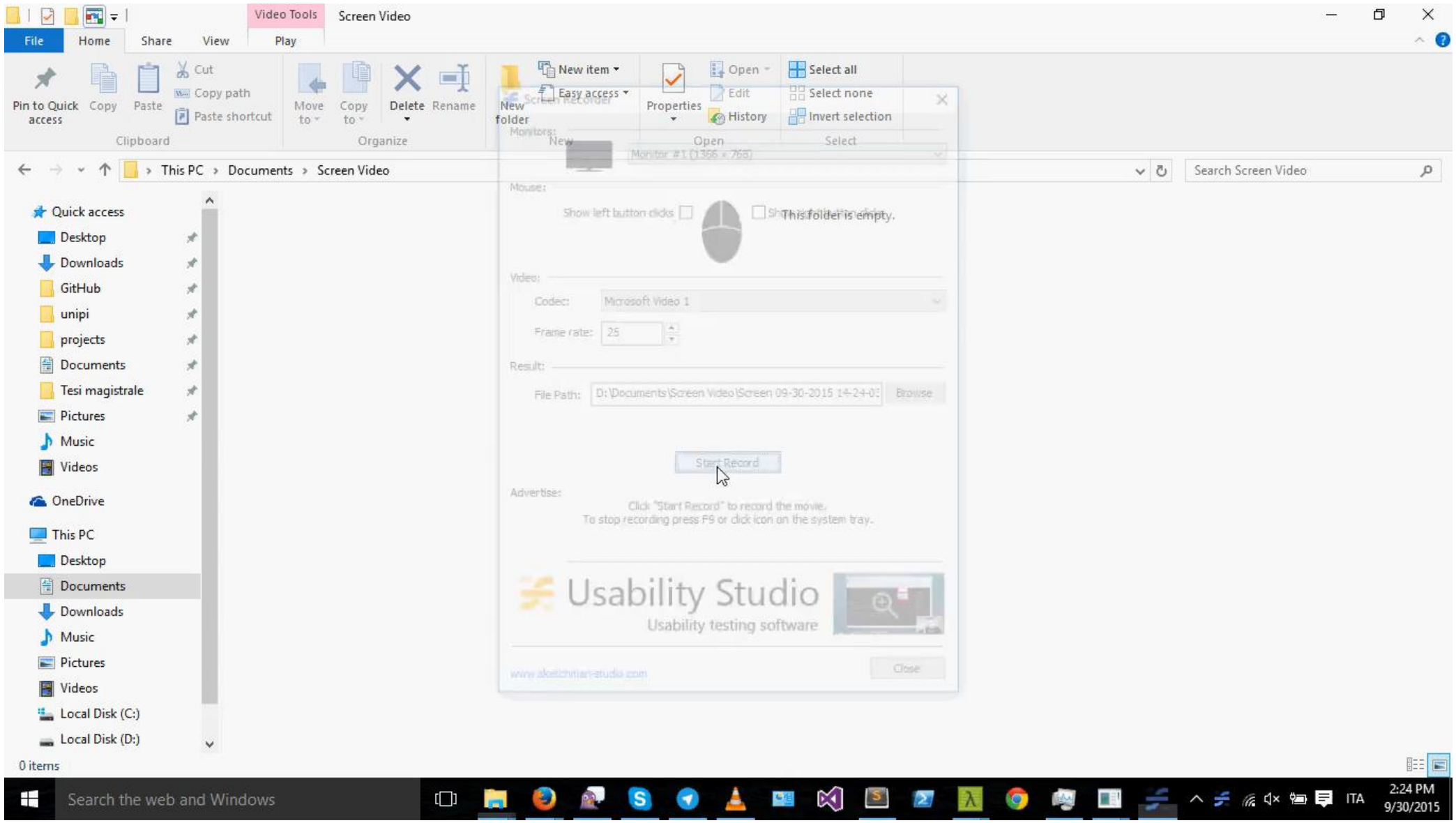




The robot is able to follow the target thanks to the neuronal filtering of the retinal output.

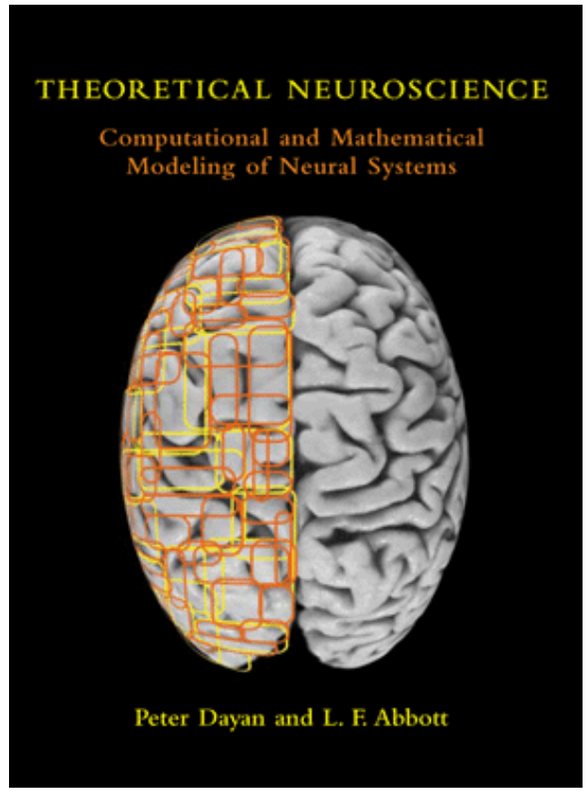
SpiNNaker visual tracking

First attempt at performing visual tracking with the iCub simulator and SpiNNaker.



Neuromorphic computing resources

Computational neuroscience:
- “Theoretical Neuroscience: computational and mathematical modeling of neural systems” by Peter Dayan and Larry Abbott
- Computational Neuroscience on Coursera



NEST:
- www.nest-simulator.org

SpiNNaker:
- spinnakermanchester.github.io
- apt.cs.manchester.ac.uk/projects/SpiNNaker

Neurorobotics Platform:
- neurorobotics.net

Others (related):
- l.vannucci@sssup.it